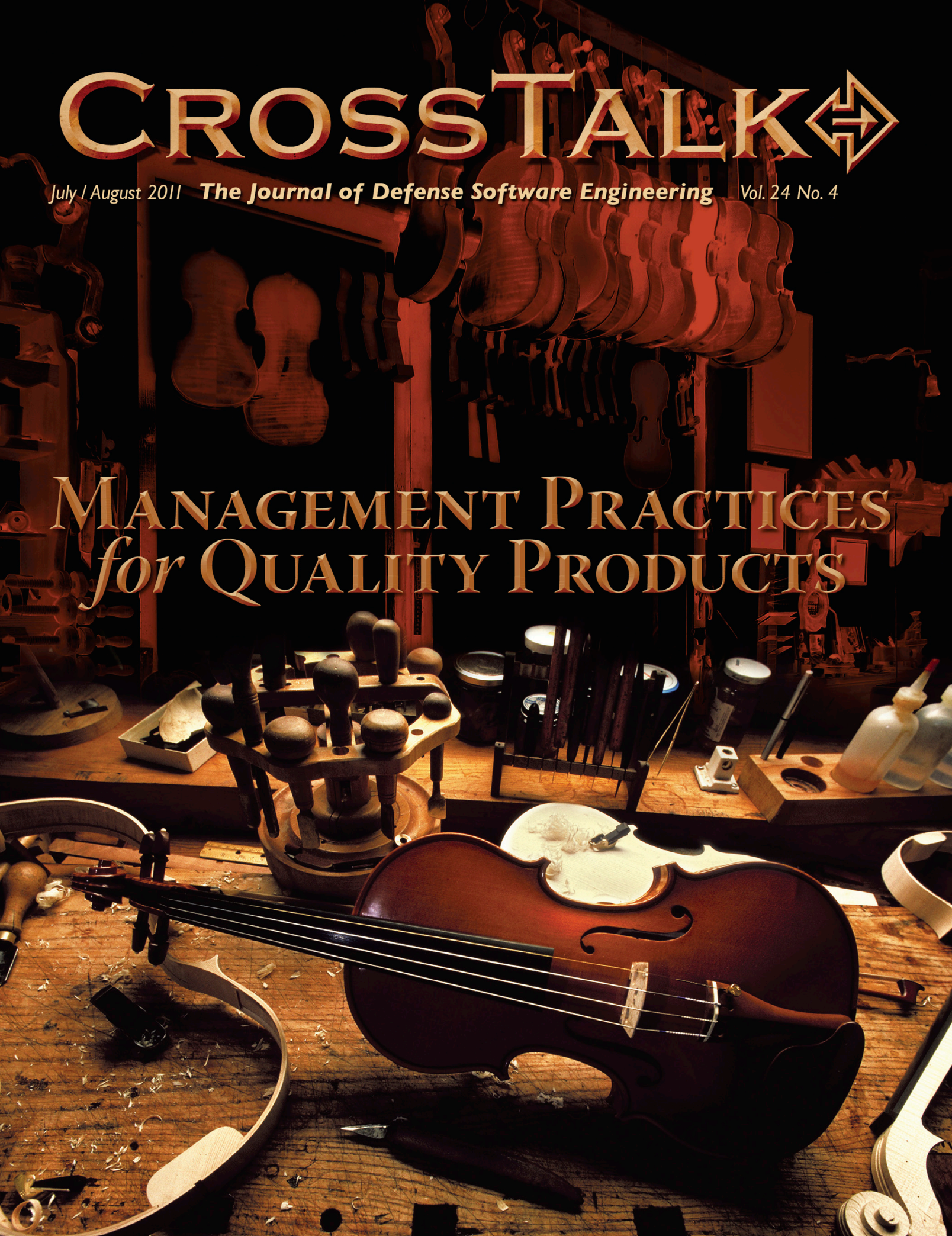

CROSSTALK
July / August 2011 The Journal of Defense Software Engineering Vol. 24 No. 4
MANAGEMENT PRACTICES
for QUALITY PRODUCTS

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUL 2011		**00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk The Journal of Defense Software Engineering. Volume 24, Number 4, July/August 2011	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

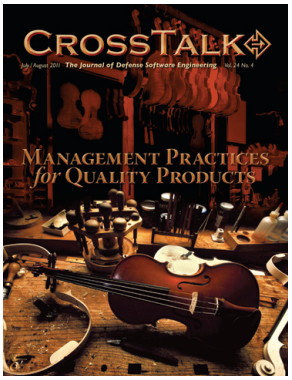
14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**40**	

CONTENTS

Departments



Cover Design by Kent Bingham

Management Practices for Quality Products

CrossTalk

OUSD(AT&L) Stephen P. Welby
NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Advisor Kasey Thompson
Article Coordinator Lynne Wade
Managing Director Brent Baxter
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-775-5555
E-mail stsc.customerservice@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the Under Secretary of Defense for Acquisition, Technology and Logistics (OUSD(AT&L)); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Defense (DHS). USD(AT&L) co-sponsor: Deputy Assistant Secretary of Defense for Systems Engineering. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Program Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of **CrossTalk** providing both editorial oversight and technical review of the journal. **CrossTalk's** mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the **CrossTalk** editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. **CrossTalk** does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with **CrossTalk.**

Trademarks and Endorsements: **CrossTalk** is an authorized publication for members of the DoD. Contents of **CrossTalk** are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the **CrossTalk** webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>. ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CrossTalk would like to thank NAVAIR for sponsoring this issue.

Management Practices for Quality Products

Management practices leading to the delivery of quality products are affected first and foremost by the responsibilities associated with the roles of the managers in an organization. By this I mean those responsibilities should be pushed further down into the organization to have an affect on the plans and the quality of the products delivered.

Redefining management means that it can no longer come from just the top of an organization. Current senior managers of programs and projects must extend the focus of being a manager down to the working level of a project. Each engineer becomes a manager of the work done. In teams, these manager-type engineers come together to plan the work given to them by the senior managers. This will result in a team plan based upon recent experience and historical data. This type of team environment produces plans that are both aggressive and realistic. Plans are granular so that progress will be tracked in hours and days versus classic plans that are tracked in weeks and months.

With this low-level management structure in place, senior managers can become better leaders. They are better equipped to support and protect their teams. For example, when new requirement changes come along, senior managers go to their team of manager-engineers, get estimates, and then go back to customers with choices based upon data.

Quality means different things to various individuals. Fundamentally, quality is about delivering products to the customer that meet their functional needs. Beneath that, the product delivered must be maintainable, reliable, and useable. To achieve these quality requirements, projects must manage the mistakes made while the product is developed. When quality is managed poorly–or not at all–the results include schedules that blow up and related major cost overruns for funding used to pay for additional time spent working and reworking the product.

Years ago when I began doing process improvement, I established a milestone for success. It was when I observed management asking about quality every time they asked about cost and schedule. What I have come to realize through the realization of spreading the definition of management down into the teams, is that this has already happened. Engineers have also taken on roles of management as they plan and track their work. Senior managers have become better leaders as they have allowed management responsibilities to move down into the teams doing the work. They better understand the quality and look for it in the products delivered as well as in the processes used.

Based upon the management approach of teams managing their own work comes the question of what really makes quality happen. Product teams and the managers above them are first given the understanding that people who do the work will make mistakes. Simply put, managing quality means managing mistakes. This happens when those mistakes are discovered and corrected as soon as possible. Earlier attention to product quality by itself dramatically contributes to cost control and staying on schedule. Increasing the number of places for removing mistakes and doing so earlier in the process allows teams to plan their work in a way that supports the delivery of the highest quality products on cost and within schedule.

What I have described here are some parts of the approach we have applied at NAVAIR. It is based upon the original work of Watts Humphrey and SEI. It is known as the Team Software Process. We have used this approach with many software teams and non-software teams with great success.

The old saying, "Better-faster-cheaper, pick any two," does not hold up when better management practices and attention to quality are applied. You can have all three when you enable the teams doing the work to manage themselves in a domain of planning based upon data, tracking product tasks with data, and early detection of mistakes so that quality products are delivered on budget and on schedule.

Jeffrey Schwalb
Naval Air Systems Command

Watts Humphrey

The Father of Software Quality (1927-2010)

Carnegie Mellon Software Engineering Institute

When Watts Humphrey arrived at the SEI in 1986, he made what he called an, "outrageous commitment to change the world of software engineering."

By all accounts, he succeeded. Known as the "Father of Software Quality," Humphrey dedicated his career to addressing problems in software development including schedule delays, cost increases, performance problems, and defects. In 2005, Humphrey received the National Medal of Technology, the highest honor awarded by the President of the United States to America's leading innovators.

"He was a wonderful leader and a wonderful man. He set forth an energizing goal and an inspiring mission that we all wanted to be a part of," said Anita Carleton, director of SEI's Software Engineering Process Management program, who was initially hired by Humphrey.

An Outrageous Commitment

When he arrived at SEI after working for nearly three decades at IBM, he came with a vision: Software could be managed by process.

"Changing the world of anything is an outrageous personal commitment. I felt it needed to be done. I knew I couldn't do it alone," Humphrey explained in a 2010 interview.

"We all understood the importance of things such as version control, configuration management, and methodology, but I don't think anyone knew how to put those into a transferable form," said Larry Druffel, SEI director and CEO from 1986 to 1996.

Working with a team, Humphrey identified characteristics of best practices in software engineering that began to lay the groundwork for what would become the CMM® and eventually, the CMMI®.

After being named the first SEI Fellow–an honor given to individuals who have made an outstanding commitment to the work of SEI–Humphrey focused on the development of the Personal Software Process℠ (PSP) and Team Software Process℠ (TSP) initiatives.

The Beginnings of PSP and TSP

Jim Over, who now leads the TSP initiative at SEI, said Humphrey had begun his work in bringing discipline to the individual software engineer–the basis for the PSP–long before his appointment as an SEI Fellow.

Humphrey first tested his theories on a process that he developed for managing his personal checking account. Next, he tested them on the personal software development process by writing more than 60 small programs in Pascal and C++, Over explained. Humphrey then began working with organizations to pilot this new personal process for software engineers.

Not long after, Humphrey published his first PSP book, "A Discipline for Software Engineering," and developed a course for software engineers. Over, who enrolled in the first PSP course offered at Carnegie Mellon, said it changed his career.

"When you learn how to properly measure your own performance and analyze the result in order to improve, you get real, lasting, behavioral change that leads to performance gains and improvement," Over explained. The class, he said, went from underestimating its work by about 40 % to being within a few points under or over on each assignment. "We had a 10 times reduction in the number of defects that escaped to the unit testing phase by the end of the course. These results were unbelievable. If I hadn't been there, I would not have thought this possible." After the course, Over began working with Humphrey to transition PSP and TSP into software engineering practice.

Humphrey faced many naysayers, Druffel recalled. With each critic, he would listen and adjust his approach, but never once did he give up on the idea that he could teach software engineers the skills that they needed to track their own work, adhere to plans, and develop defect-free software. After PSP was established, Humphrey applied those same concepts to engineering groups as part of TSP. Today, TSP has been adopted by leading software organizations across the globe including Intuit, Oracle, and Adobe.

"What Watts brought is an acceptance of the discipline of software engineering," Druffel explained. "He was working on these ideas when he left IBM in 1986. When he died in 2010, he was still working on related concepts. That's persistence."

Quantitative Project Management Framework via Integrating Six Sigma and PSP/TSP

Sejun Kim, BISTel

Okjoo Choi, KAIST

Jongmoon Baik, KAIST

Abstract: Process technologies such as Personal Software Process℠ (PSP) and Team Software Process℠ (TSP) provide a good foundation for Six Sigma applications in business. Business approaches using Six Sigma provide methods for process improvement and analysis to achieve the goals of the PSP/TSP. This article discusses a framework with which software engineers and project managers can quantitatively manage software projects for improving the processes by applying Six Sigma in conjunction with PSP/TSP.

Introduction

The advent of CMMI® has helped software engineers and project managers understand the principles and approaches of software process improvement [1, 2]. Many people believe if an organization achieves a higher CMMI maturity level, a higher performance of the software processes follows. The performance they achieve depends on their executions of each CMMI Process Area (PA). CMMI is a framework that helps improve software product quality and productivity but not processes. CMMI describes the characteristics of processes but not the processes themselves. In other words, CMMI just includes "what." There have been difficulties in increasing productivity with these models because "how" is not within the scope of the CMMI. Thus, software engineers and project managers know the goal of their project, but they do not know how to implement each procedure of the CMMI PAs or have the means to improve the processes for their goal. SEI has introduced "how to" technologies for CMMI at the individual and team level with PSP and TSP. There are also several "how to" technologies, appraisal methods, PSP/TSP, and measurement and analysis tools, which are foundations of the solution for high performance of software process [3].

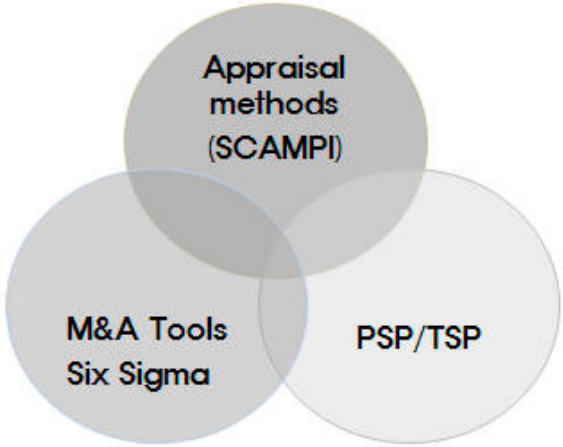


Figure 1: The "How to" Technologies for high performance

In this article, we focus on the integrated use of Six Sigma and PSP/TSP for higher performance of software processes and project management. PSP helps individual developers improve their performance by bringing a discipline to the way that they develop their software [4]. TSP provides software engineers and managers with a way to establish and manage their team to produce high quality software within a given schedule and budget [5]. Six Sigma is a quality improvement approach to enhancing an organization's performance by using statistical analytic techniques [6]. It provides the quantitative analysis tools necessary to control process performance.

Many organizations that endeavor to improve software processes often find themselves integrating many approaches to achieve that improvement. Integrating Six Sigma and PSP/TSP can enable software engineers and project managers to analyze PSP/TSP data and to systematically improve process performance at an organization level. To do this, we map Six Sigma tools to each PSP/TSP process in order to show what Six Sigma techniques can be applied to the data of a given PSP/TSP and suggest Six Sigma practical usage guidelines to support process improvement activities at an individual and team level. However, there are a few analysis tools, such as process dashboard [7], Hackystat [8, 9], and PSP Assistant [10], and systematic process control functionality metrics collected in PSP/TSP activities. This article suggests Six Sigma and PSP/TSP tools that we have developed and proposes a framework for integrating those tools based on a knowledge-base repository. Thus, we can create a quantitative project management methodology by integrating Six Sigma and PSP/TSP based on a knowledge-base repository.

Integrating the "How to" Technologies

Six Sigma supports software process improvement in PSP/TSP activities and helps organizations to achieve process improvement goals at an organizational level. PSP provides individual level project data containing information from what should be done to what has been done. Then, TSP is used in order to extend the collected data into a team view. Despite the fact that TSP activities are mostly based on the PSP results, PSP and TSP should be handled in different ways according to their different points of view. This means that the way of utilizing Six Sigma for each process should be different.

Since most of the PSP metrics and quality assurance activities are embedded in TSP, an adequate understanding of PSP is necessary. Six Sigma provides PSP/TSP with various tools for detecting special causes of variation, evaluating the impact of process changes, and improving process performance at an organizational level. Figure 2 presents the relationships among the "how to" techniques at an organization level. Using Six Sigma tools, individual-level data gathered from developers, following PSP0 through PSP3, is managed and analyzed. In addition, based on the PSP data, which is transformed into TSP data at a team level, TSP establishes a defined process foundation and generates useful data that can be analyzed using Six Sigma tools. The analysis results from Six Sigma provide methods for analyzing collected data in PSP/TSP and leads to individual and team level (further, organizational level) performance improvement through effective decision making.

Six Sigma provides various statistical and non-statistical tools in order to support effective decision making in the process of developing software. Mapping Six Sigma tools to each PSP/TSP process helps software engineers and managers understand how to use Six Sigma analysis techniques in conjunction with PSP/TSP data. Mapping the Six Sigma and PSP/TSP process, shown in Table 1, describes the statistical analysis and decision-making support tools of each PSP/TSP phase and its purpose. Since PSP/TSP activities are performed in several cycles, more Six Sigma tools can be applied in later cycles. The main issues of the mapping between Six Sigma and TSP actually rely on information gathered from the PSP activities. In other words, it is important to define what and how to extend the individual data to the team level information. To do so, we have defined several steps by tailoring the TSP launch process in [5]: Strategy, plan, risk, assessment, review and postmortem.

An Integrated Framework for Six Sigma and PSP/TSP

Six Sigma and PSP/TSP Tools

We suggest frameworks and implemented relative tools of Six Sigma and PSP/TSP, Six Sigma Project Management Tool (SSPMT), JASMINE, and ALADDIN, respectively [11, 12]. SSPMT is a web-based Six Sigma project management supporting tool that supports Define, Measure, Analyze, Improve, and Control (DMAIC) and Design for Six Sigma methodologies. Using the project initiation data and PSP/TSP data gathered from JASMINE and ALADDIN, SSPMT performs each step of DMAIC and provides analytic results. JASMINE and ALADDIN are web-based PSP and TSP project supporting tools, respectively. Since TSP mostly gathers information from the PSP activity results, most of the process works are done by using JASMINE. JASMINE collects an individual developer's work product information such as Source Lines of Code (SLOC), fault counts, and so on. When a system is developed using Eclipse, it provides plug-in that automatically collects bug occurrence information per compiler. ALADDIN recollects the individual level project data and categorizes it at the predefined team level for further organizational decision making. Although the tools interact with each other, since they use individual data repositories, they are not fully integrated from the management point of view. Our intuition is that the decentralized database reduces the capability of managing the output of each process and further quantifying decision-making variables or measurements.

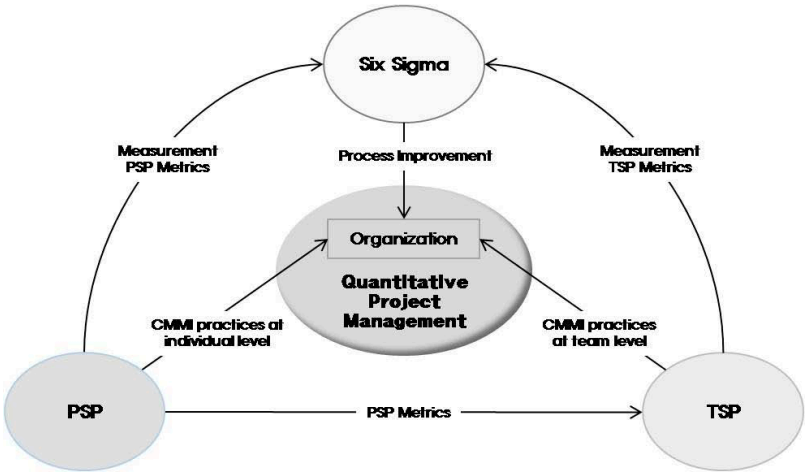


Figure 2: Integrating the "How to" technologies

TSP		PSP		Six Sigma Tools	Purpose
Strategy	Project objective and strategy	PSP0	Planning	Correlation analysis, Regression analysis	To estimate development time of new program
	Team Goals and Roles		Development	Dotplot, pareto analysis	To analyze the percentage of defects by defect type
Plan	Overall Plan		Postmortem	Correlation analysis, Regression analysis	To estimate size and development time of new program
	Quality Plan	PSP0.1	Planning		
	Job Allocation Plan			Correlation analysis, Regression analysis	To estimate Added and Modified value for new program
Risk Assessment	Project Risk Analysis		Development	Multiple regression analysis	To analyze Added, Reused, Modified value distribution of current program and to estimate LOC for new program
			Postmortem	The same tools as in postmortem phase in PSP0	
Review	Report Generation	PSP1	Planning	PROBE(PROxy-Based Estimating)	To estimate new program size and development time
			Development	The same tools as in development phase in PSP0	
	Management Review		Postmortem	2-sample t-test	To determine whether estimations has been correct by comparing between Plan Size/Hour values and To Date Size/Hour values
Postmortem	Launch Postmortem	…			
				Process Charter Process Evaluation Sigma Calculator	
	Quantitative Project Management			…	To estimate using quantitative indicator

Table 1: Mapping table of Six Sigma and PSP/TSP

As the PSP/TSP process continues, Six Sigma quantifies the results of the processes by using various tools in order to provide decision-making support. The detailed procedure of this process, using the existing tools (SSPMT, JASMINE, and ALADDIN), can be described as follows:

1. Initiate the PSP/TSP process using JASMINE and ALADDIN.
2. Store the PSP/TSP data in the data repository of SSPMT.
3. Analyze the PSP data and report individual level process performance improvement and decision-making issues using the SSPMT.
4. Organize the PSP data into the predefined team in order to support TSP data analysis.

5. Analyze the TSP data and report the team level process performance improvement and decision-making issues using the SSPMT.
6. Generate the results of both the individual and the team level decision-making report.
7. Keep track of and provide feedback to the next cycle.

First, the PSP and TSP are performed by following their own process using JASMINE and ALADDIN. Then, the overall data is stored in the data repository, which is in the SSPMT database. (We will discuss the repository in the next section in detail.) ALADDIN collects the results, gathered from JASMINE, and combines them in the form of teams that were predefined in TSP team building in order to support team level decision makings and analyses. Finally, the SSPMT generates the decision-making report according to the data and its analysis.

According to the procedures above and the mapping table shown in the previous section, the suggested architecture supports not only PSP/TSP activities, but also their relative analysis results and decision-making issues at an individual level and a team level. In addition, the results of the statistical data analysis help the project managers and software engineers to readily make various decisions, for example, changing the management. It is also easier to manage each process's data concurrently by integrating the data repositories.

However, since the tools use individually distributed data repositories, the measurements, relative matrix, and results are not managed concurrently and there is also needless storage waste. For example, since most of the data analysis results of the TSP are based on the PSP data, the TSP tool itself does not need to be inputted again and/or the data restored. Thus, it is better to directly store the necessary measurements and minimize duplication.

Knowledge-based Data Repository

According to the facts stated above, the suggested architecture is based on the integration of the database of the three methodologies. Since it manages the overall data of each process results by integrating the database, a more quantitative and integrated process and project management can be provided.

In order to integrate the supporting tools of the three processes, we provide a knowledge-based database. As shown in Figure 4, the database architecture consists of three data repositories as follows:

1. Master Data Repository:
   Project Master Data: contains project initiation data, such as baseline, team members, resources, schedule, measurements, process mapping information, etc.
   Process Master Data: contains setup information for Six Sigma and PSP/TSP processes (e.g., DMAIC, DFSS of Six Sigma).
2. Instance Data Repository: stores each process's empirical data (measurement) produced by each tool (e.g., Six Sigma instance data, PSP/TSP instance data).
3. Analytic Data Repository: stores analytic results of instance data using Six Sigma data analysis.

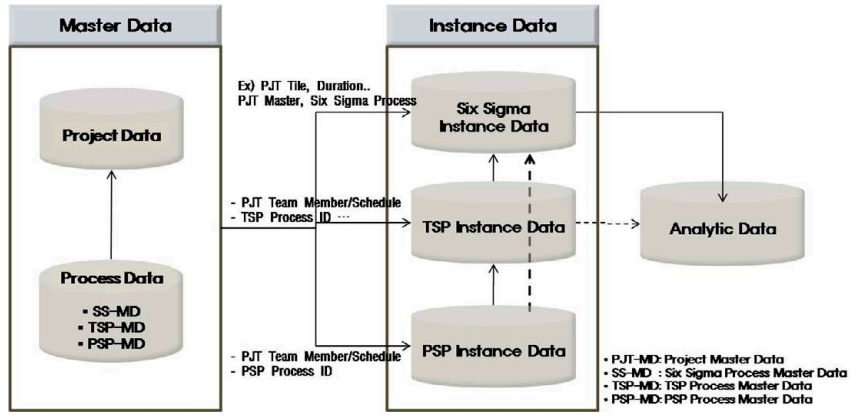


Figure 3: Data Repositories

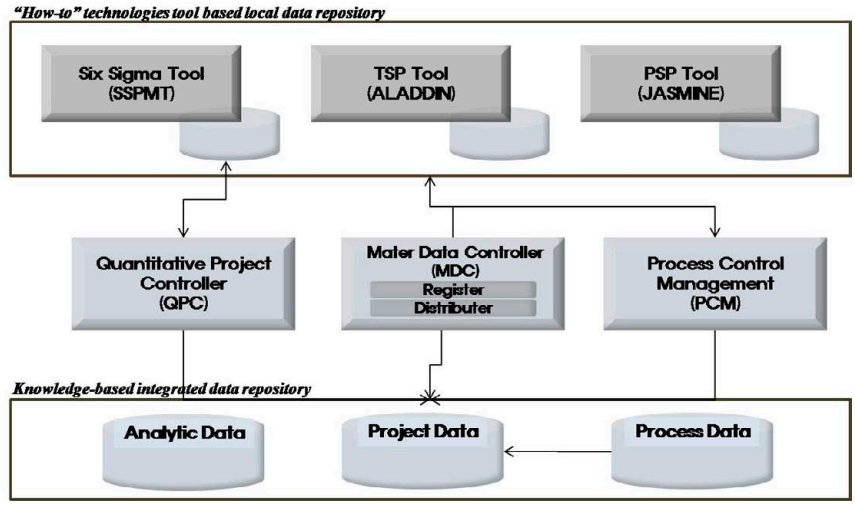


Figure 4: Integrated framework for data management

The master data repository, which contains project initiation data and process setups, provides the basic information of a project and its process to the instance data repository. For example, the Six Sigma framework and team information can be used in Six Sigma instance data and TSP data, such as process ID and process name, and PSP data, such as individual member information, PSP process ID, and name, can be used in TSP instance data, and PSP instance data, respectively.

Based on the data collected, PSP activities are performed and TSP extends the results to a team level using the team member information and process ID. While PSP/TSP tools perform their process, Six Sigma instance data repository collects the results in order to perform the data analysis that is used to support decision making. Finally, the analytic results are stored in the analytic data repository.

Data Management

In order to support the data repository framework, we implemented a central data management application, QPC, MDC, and PCM for managing and analyzing the metrics from the PSP/TSP.

The register in the MDC, which is the central data management application, receives the process and project master data (e.g., project title, budget, schedule, and so on) from the managers. The distributer then distributes the information to each tool.

According to the data, JASMINE performs the PSP activities and ALADDIN combines the results at a team level and performs the TSP activities. The SSPMT then analyzes the measurements collected from the tools and provides decision-making reports. During the Six Sigma analysis, the PCM continually monitors the processes and feedbacks the analysis results to the organization in the form of a report or e-mail. QPC transfers the analysis data of the SSPMT to the analytic data repository, and if required, to other data repositories.

Supporting Decision Making

An Integrated Process

In this section, we suggest a practical guideline showing how the framework can be used by providing an example process of the suggested framework. Figure 5 shows the overall process of the suggested framework. The process can be categorized into three layers: administrative, project, and organization.

First of all, the administrative layer works from the administrator's point of view. In this layer, the project manager registers the project basic information and its process information and maps the process and the project. As mentioned in the previous subsection, all the project initiation information can be registered with the MDC. If a similar project exists, the project manager revises it and uses a new project template.

In the project layer, PSP and TSP processes are performed according to the predefined mapping information. As shown in Figure 5, each project team member first performs his or her individual role by following the PSP activities PSP0 through PSP3. Then, they gather the individually performed outputs (e.g., SLOC, fault count) in order to extend it to the team level using the TSP based on the team information and TSP process ID gathered from the administrative layer. Data collected in this layer is stored in the instance data repository.

In the organization layer, using the individual and team level information gathered from the previous step, the Six Sigma process is performed using appropriate tools based on the predefined framework. Using statistical and non-statistical analysis, Six Sigma provides analysis results at the project, individual, team, and organizational levels. According to the results, the SSPMT provides an analysis report that quantifies the overall results and enhances organizational decision making. By using the PCM, project managers can also monitor whether each project is going well. Finally, the feedback based on the overall results and relative reports can be used to improve the whole development lifecycle and further organizational improvement.

Quantitative Project Management

Based on the suggested integrated framework and processes, we can collect individual (x_{psp}) and team (x_{tsp}) data through the PSP/TSP processes at the project layer. We can also elicit a set of metrics (x_{tsp}) from individual (x_{psp}) and team (x_{tsp}) data in the TSP phase. The data is analyzed using Six Sigma tools at each project layer and organization layer. Then, the Quantitative Management Indicator (QMI) absorbs the analyzed data (y_{target}) and determines if it satisfies the organizational goal. If so, the process continues the same as at present. If not, individuals and the team will receive feedback indicating what and where the problems are. Then, the process will be changed or fixed according to the issues and the process will be repeated with newly collected metrics until the QMI confirms that the goal is satisfied.

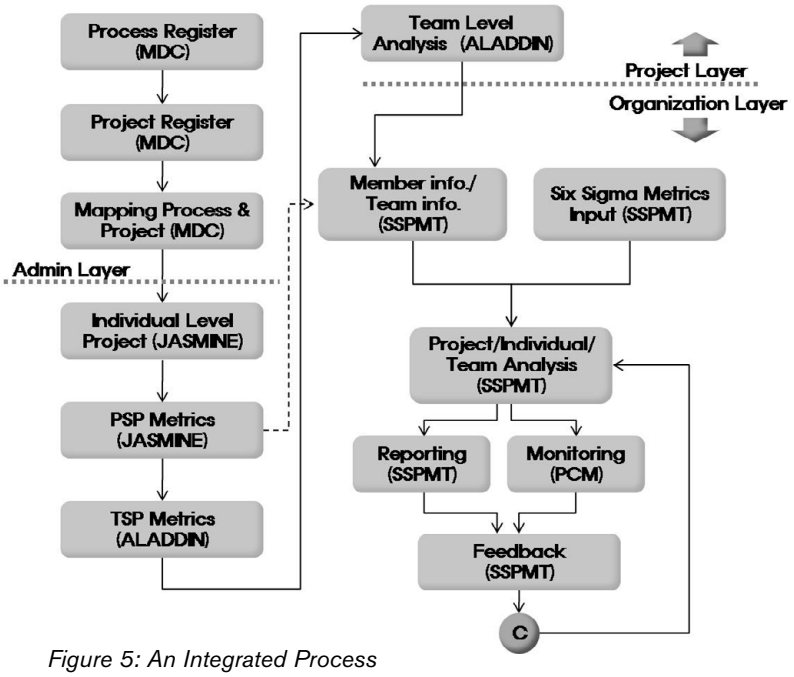


Figure 5: An Integrated Process

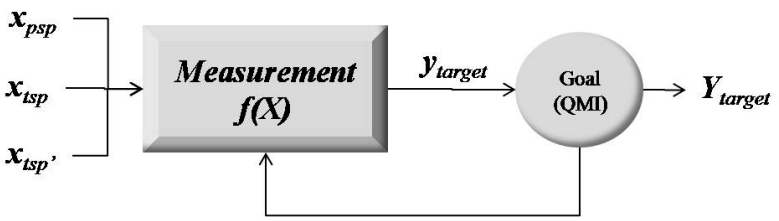


Figure 6: Quantitative Project Controller

Using the QMI, it is possible to directly relate the individual/team level data and related metrics to the organizational goal. Since it indicates the locations of the cause of the disconfirmation, organizations can reduce the cost and change the schedule of the process execution. As a result, by applying the QMI at each PSP/TSP phase within Six Sigma's quantitative measurements, organizations can deliver their products with the desired quality, which will lead to customer satisfaction.

Conclusion

This article focused on supporting quantitative decision making for process performance during software development projects. It is proposed to seamlessly integrate Six Sigma and PSP/TSP tools using a knowledge database. Thus, an organization can continuously improve its process based on empirical and analytic data and move to a higher CMMI level. In the future, we expect to develop more accurate metrics for quantitative project management of each domain and project guidance.

Acknowledgements

This research was supported by The Ministry of Knowledge Economy, Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency). (NIPA-2009-(C1090-0902-0032))

Disclaimer

®CMMI and ®CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
℠PSP and ℠TSP are service marks of Carnegie Mellon University.

REFERENCES

1. Mark C. Paulk, B. Curtis, M. B. Chrissis and C. V. Weber, "Capability Maturity Model for Software, Version 1.1", Technical Report CMU/SEI-93-TR-024, ESC-TR-93-177, Feb, 1993.
2. CMMI, "Capability Maturity Model Integration" Software Engineering Institute, CMU, 2002.
3. CMMI Made Practical, "Implementing CMMI for High-Performance", London, April, 2009.
4. Watts S. Humphrey, "The Personal Software Process", Technical Report, CMU/SEI-2000-TR-022, ESC-TR-2000-022, Nov, 2000.
5. Watts S. Humphrey, "The Team Software Process", Technical Report, CMU/SEI-2000-TR-023, ESC-TR-2000-023, Nov, 2000.
6. Thomas Pyzdek, "The Six Sigma Handbook: The Complete Guide for Greenbelts, Blackbelts, and Managers at All Levels, Revised and Expanded Edition, McGraw-Hill, 2003.
7. <http://www.processdash.com/>
8. Johnson, Philip M.,"Project Hackystat: Accelerating adoption of empirically guided software development through non-disruptive, developer-centric, in-process data collection and analysis", Technical Report csdl2-01-13, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, November 2001
9. <http://code.google.com/p/hackystat/>
10. R. Sison, et al, "Personal Software Process (PSP) Assistant", In Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)
11. Y. Park, H. Park, H. Choi, and J. Baik, "A Study on the Application of Six Sigma Tools to PSP/TSP for Process Improvement", In Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06).
12. P. Zedan, H. Park, and J. Baik, "A Six Sigma Framework for Software Process Improvements and Its Implementation", In Proceedings of APSEC 2007, Dec 5-7, 2007.

ABOUT THE AUTHORS

Sejun Kim is an Engineer at BISTel, Seoul, Korea. Kim earned a master's degree in computer science at the Korea Advanced Institute of Science and Technology (KAIST) and a bachelor's degree in computer science at Kwangwoon University. Kim was involved in the Software Process Improvement Center project and has developed the SSPMT, CMMI-Six Sigma Guideline, and Reliability-Six Sigma Guideline. His research interests are software process and quality improvement, especially focused on Software Six Sigma and Software Testing.

KAIST
335 Gwahak-ro (373-1 Guseong-dong), Yuseong-gu, Daejeon 305-701, Republic of Korea
Phone: +82-42-350-3356
E-mail: sejunkim@kaist.ac.kr

Okjoo Choi is a research assistant professor at the department of computer science at KAIST. Choi earned a bachelor's degree, a master's, and Ph.D. in computer science from Sookmyung Women's University, Seoul, Korea. Before she joined KAIST, she had worked at ERP consulting services, Oracle Korea Ltd. as a consulting technical manager from 1996 to 2009 and LG Electronics Production Engineering Research Center an assistant research engineer from 1990 to 1996. She is currently involved in projects related to software reliability for embedded weapon systems.

KAIST
335 Gwahak-ro (373-1 Guseong-dong), Yuseong-gu, Daejeon 305-701, Republic of Korea
Phone: +82-42-350-3356
E-mail: okjoo.choi@kaist.ac.kr

Jongmoon Baik is an associate professor at the department of computer science at KAIST. Baik earned a bachelor's degree in computer science and statistics from Chosun University and a Master's and Ph.D. degree in computer science from University of Southern California. Before he joined KAIST, he was a principal research scientist at Software and Systems Engineering Research Laboratory, Motorola Labs from 2001 to 2005. He is also an adjunct faculty member for the Masters of Software Engineering program at Carnegie Mellon University.

KAIST
335 Gwahak-ro (373-1 Guseong-dong), Yuseong-gu, Daejeon 305-701, Republic of Korea
Phone: +82-42-350-3356
E-mail: jbaik@kaist.ac.kr

Driving Major Change

The Balance Between Methods and People

David M. Moore, COL, U.S. Army, Project Manager Battle Command

Portia Crowe, U.S. Army, Chief Engineer PM BC-Strategic

Robert Cloutier, Ph.D., Stevens Institute of Technology

Abstract: Successful system of systems interoperability includes a disciplined and responsive system engineering process that focuses on both near-term deliveries to new and current software baselines and longer-term development that sets conditions for enhanced warfighter effectiveness. The foundation of this success relies on flexible and rapid development methodologies and the creation and sustainment of a collaborative social environment by which various communities unify to provide capabilities in a common framework. In the context of new strategy development, the intent of this article is to describe the challenges of implementing an innovative and collaborative environment in the context of scaling an agile system engineering method to a large combined effort.

"Technical problems we can solve; social challenges are much harder." These words of wisdom stated by our Project Manager Battle Command (PM BC) Technical Manager have proven true many times over. The core meaning of this statement is that engineers tend to focus on innovative methods and best practices as a means to increase productivity, reduce defects, increase cycle time, et. al. The most critical processes and methods to success really involve unifying and sustaining a productive social component – a good team with clarity of mission, unity of purpose, and organized to clear objectives.

The backdrop for what has become more of a social endeavor than a technical shift is the Battle Command (BC) "collapse" strategy. The Army has been developing unique and essentially stove-piped digital command and control solutions for many years. Nearly every specific staff function (artillery, air defense, aviation, etc.) has built a unique system for its sole purpose. While it must be noted that Army doctrine drove this design, the sum effect is that our unit commanders and staff are separated by their information systems. The collapse strategy implements a material design approach that brings the Army's family of uniquely distinct tactical functional applications with unique data storing and sharing mechanisms and collapses these systems towards a consolidated software product line. The desired benefits of the collapse strategy beyond the operational warfighter value included reduction in software development and hardware procurement costs.

To launch a strategy about a core product, the BC leadership first had to create conditions for unified effort as well as establish overarching system engineering processes to control progress and gain irreversible momentum. Irreversible momentum is achieved by establishing annual build plans and driving immediate redirection of effort towards these near term goals. Overarching system engineering processes at the PM BC level had to be established in parallel to subordinate project plan adjustments. As part of the BC effort, agile methodologies were adopted and built into the broader organizational culture.

A Shared Innovator's Environment

An aggressive rapid approach is a key measure of keeping our capability relevant with deployed warfighters. One barrier to innovation is a program having centralized control over the development environment. Innovation is more broadly adopted when all can participate with a degree of independence and recognition of shared value by unity of effort.

The main effort of the BC strategy was to build on the most promising software framework within the BC portfolio. This strategy achieved immediate gains but also advanced a limitation that this architecture was proprietary. To mitigate this limitation, PM BC negotiated for government purpose rights (GPR) within the next year. This allowed all developers to maintain the BC software framework. This approach also created the most internal social friction and demanded very deliberate and significant system engineering. Subordinate Project Managers (PMs) and their contractors who needed to shift to this new architecture needed significant training. PM BC continues to mediate between contractors to maintain as much momentum as possible and to keep on the annual build schedules. In time and with the release of GPR, the social friction associated with this course of action should diminish but success has demanded significant leader interaction to maintain support and keep the system engineering process on track.

Similar to each subordinate PM maintaining a thick client system, each PM was also developing unique web service frameworks and a unique presentation layer. To unify this unacceptable condition, the PM chose a new product development

effort and built a common web service environment usable at the tactical level that extended the commander's collaborative reach to anyone with a computer and browser ("BC Web"). Select functions are targeted for this environment with the goal of eventually building collaborative capability in a web service environment. To maximize the ability to team with a broad range of development partners, PM BC chose a government open source framework that already had momentum in the Intelligence domain. This choice reinforced unified effort within the Army and created the conditions for any command to build with a program of record in a collaborative development environment.

With the intent to collapse disparate BC thin client systems, a third-party environment became necessary to allow other developers to create, or re-create, their capabilities using a set of standard tools and guidelines in a common framework. The BC Web thin client team stood up an environment that includes a third-party software development kit, third-party widget test and development area using Defense Information Systems Agency (DISA) Rapid Access Computing Environment, authorization and authentication, widget security checklist, and widget and training style guides that collectively provide an integrated secure single environment from development to deployment. The team was able to completely stand up this environment within six months in accordance with the Army CIO/G6 guidance and policies for a common operating environment [1]. One of the main efficiencies of using a common operating environment is that Battle Command, Distributed Common Ground Systems-Army, and now the DISA Joint Command and Control initiative will provide a common core framework and capabilities for broad applicability for enterprise and tactical users, and will allow for optimal sharing of information, infrastructure, and development costs using the Ozone Widget Framework.

Through innovation and placing value on a new product, we are trying to motivate users and developers to create, share, and enhance capabilities and allow for efficiencies in products, services, and processes at a monumental pace, getting users what they need. Through these efforts, our users are seeing similar functionality to common web features (i.e., social networking, mapping features, app store concept). An aggressive strategy, modern technologies, and warfighter needs have imposed a new business model that requires an innovative environment that allows for growth, rapid development and lean testing, integration, and deployment of capabilities. Chris Jones, a widget developer, states, "As a developer, the use of the standard set of tools increases productivity, enables rapid development and deployment of capabilities, and ensures that I maintain the rigors of software governance, test, and validation provided within the environment of BC Web."

Methodology Used

The traditional acquisition process used to develop military technology is not aligned with the speed, agility, and adaptability of new IT capabilities in today's information age [2]. To provide speed of delivery of capabilities to warfighters, we choose to implement an Agile Systems Engineering (ASE) approach that encompasses agile principles as well as brings agility to the entire lifecycle process. Parallel efforts of development, testing and integration with short iterations while stacking priorities are part of the agility structure implemented. The beneficial impact of agile systems engineering is to work smarter and provide immediate benefit and value to the users. It is a highly collaborative method that needs the stakeholders to work together to be successful. The agile systems engineering method values customer interaction and collaboration over process [3].

Future Trends in Systems Engineering
Platform to enterprise (customer emphasis)
Dominant prime to strategic teaming (Execution internal & external)
Compressed delivery schedules
Increased reliance on systems engineering for unknown space
Improvements in collaboration
Increased number of complex systems, emergence and rapid change
Increased customer requests for system engineering support earlier in life cycle
Increased emphasis on users and end value
Understanding of what is attainable

Table 1: Future Trends in Systems Engineering

The BC team understands the challenges of an agile approach from historical knowledge and use in other programs. However, we needed to expand this knowledge to introduce concepts and process to traditional thinkers to invoke the broader community effort. A traditional development approach starts with a defined system with specific functionality as opposed to an ASE approach where adaptive and emergent requirements and system capabilities can be undefined in the beginning and later evolve. Although agile software development is the most popular agile discipline, we needed agility across the spectrum of the program's lifecycle in a rapid and flexible manner. We incorporated ASE as a lightweight loop-back process with short and rapid cycles with priority of requirements and close user and stakeholder collaboration.

ASE offered us a way to incorporate other functions into our 30-day sprint cycle. Within one month, the 20-person contract and government team created enough momentum to demonstrate capabilities at a BC scrum (user and developer integration and feedback) event which included user stories and plans for refinement of capabilities.

Requirements from a much larger community were prioritized and the team was able to complete about 10-15 requirements a sprint. We also worked with the open Ozone community on requirements, standards, and governance. Risks were continuously monitored on a weekly basis during integrated product team meetings. Through these activities, we were able to invoke discipline in our agile processes. Testing and integration were continuously performed on each 30-day sprint build. Through the agile methodology, we had a process that started the security

accreditation process early as it is usually the longest lead time. The concurrent planning and execution of security accreditation and training modules earlier than traditional waterfall processes allowed us to provide the system to a beta unit for feedback much earlier than anticipated. A success to rapid widget development in the BC Web environment was a Communications-Electronics Research, Development, and Engineering Center research and development social networking capability that was ready for deployment for the beta unit test. The rapid development of widgets from third-party developers into a common marketplace with no middle integrator enables development of capabilities at a much faster pace with the efficiencies of using common tools. That environment is of high value to our users.

Challenges and Lessons Learned

Creating strategy is nowhere near as hard as implementing strategy. Engineering teams tend to focus on controlling processes, schedule, and risk. The value of a new strategy is only self evident to the creator. All others must be lead along the journey. Leaders and key staff must be given the means to see the vision and work towards a common objective. Solid system engineering processes and scaling up agile methodologies is hard work. Leaders who undertake strategy development must be confident that key leaders and staff who are essential to success will pick up the pace and deliver results that lead subordinates through ambiguous and challenging times.

Judicious selection of software architecture and framework are crucial choices to launch a project with momentum. This paper identifies two key elements of a strategy with two distinct start points. One leveraged a proprietary solution with near-term promise of opening the framework up while the other began open. The more open and ubiquitous an architecture is, the better. The more closed, the greater the challenge to success.

Battle Command's adoption of scrum sessions as a broad means to unify users and developers on common visualizations was essential. Scrum sessions gave a means for project managers and developers to visualize how their traditionally stove-piped software would work in the new environments. Developers were able to derive accurate requirements as a result of scrum sessions. An interesting side effect of scrum sessions was accelerated software development. By putting users and software developers together, management was sidelined. Visible angst existed as both government and commercial managers lost an element of control as these groups became excited and began to turn iterations very quickly. Getting the middleman out of the way at certain times has its benefits.

This is a people business. When change is implemented, people assess their posture against this change and will judge themselves a winner or a loser. They will get on board, actively or passively fight change, or seek a means to ride the fence, ready to shift from side to side depending on their own assessment of probability of success. Dedicated leadership at many levels is needed to overcome these dynamics. Top leaders must engage not only immediate leaders but also engineers and managers at all levels. Leaders must personally communicate the strategy, the plan, and seek feedback at every level. Sitting in the office and publishing implementing e-mails will not lead to long-term success.

At the right time, the leader must get tough. When a risk assessment warrants it, a leader must dive as deep as he can stand to draw out barriers to success. Very often these investigations will not only give technical insight into barriers to success but will also reveal subordinate interactions between government and contractors that may be barriers to success.

Do not jump to adverse social conclusions. Implementing major change is hard. It may be easy at some point to label a key leader or engineer a non-supporter and then seek methods to minimize their adverse impact. In reality, these people are most likely struggling to fit their skills and personality into the new strategy. A personnel change may be warranted if unacceptable risk persists, but this does not mean the person in charge was seeking to undermine the strategy. A leader should seek to align subordinates to their strengths if a change in strategy has marginalized an individual's value.

Nothing beats personal presence. It is a leader's responsibility to put himself in front of his subordinates when he implements change. The leader must be available to take private and public shots from his subordinates. A leader must get out, explain, and internalize how people are feeling about change. The real issues will never come in the leader's office, but brew in the cubicles of subordinates and contractors affected by change. Subordinates may not like what the leader is doing but they will always respect genuine personal engagement.

Innovation is emergent and dynamic and BC realized that it is typically a bottom-up approach in which people involvement is critical. We realized that to gain stakeholder buy-in, the team thrived off of empowerment and involvement in the methodology and process. To overcome cultural challenges, we worked with leadership for buy-in of the agile process that lead to stakeholder ownership of the process and encouraged every member of the team to participate in sprint reviews and creation of the environment. Through this process, we found innovation came naturally and was accepted more openly. The rapid and aggressive approach also brought a higher number of risks than a traditional process, so we had to adjust our tolerance for acceptance and balance it with value to our users. As more developers enter this space with unique requirements, this becomes more complex. Lessons learned also included setting expectations up front for all participants. For example, to minimize costs, the team was asked to maintain a lean attitude up front so that, collectively, we would ensure all IT dollars would provide value. We also found that culture plays a much larger role than technology and can significantly hinder or provide momentum to organizational efficiency and effectiveness. We found that common beliefs and shared logic was very important for success.

Conclusion

The BC collapse strategy is driving significant positive strides that will increase commander and staff operational effectiveness. The strategy's focus on robust, collaborative solutions places Army software development on a path to successfully supporting the warfighter in highly variable and uncertain operational environments. By breaking down system designs that have stove-piped the Army's warfighting functions, employing a human-centered collaborative approach has proven to support the way the commander and staff desire to interact. The condition PM BC seeks to create is one where the strengths of its vendors are not marginalized because of governmental barriers to effective collaboration and open competition. Adopting a commercial competitive model, characterized by rapid cycle times that quickly deliver innovation to the field, is how PM BC programs will remain relevant to the warfighter.

Technical problems can be solved; social problems are much tougher. With any change, new processes must be built or modified and then reinforced. Beneath repeatable system engineering processes and agile methodologies are people. Strategy and its supporting development processes begin and end with people. A leader must ensure his team is well trained, given a clear mission and objectives, and are resourced to execute. The BC software development mission goes one step further as our systems are used in combat. A leader will visibly display this emotional commitment to the warfighter and seek to generate and sustain this commitment in the organization. In Battle Command, this has not been difficult. The unique challenge is convincing the organization that by supporting change, the warfighter will be more lethal and survivable. Software is much about method but in the end, it is mostly about people. People drive success or failure in any organization. Active leadership at all levels drive this success.

REFERENCES

1. US Army CIO G6. Common Operating Environment Architecture. October 2010. <http://ciog6.army.mil/LinkClick.aspx?fileticket=udbujAHXmK0%3D&tabid=79>.
2. Report to Congress. A New Approach for Delivering Information Technology Capabilities in the Department of Defense. Office of the Secretary of Defense. November 2010.
3. Turner, Richard. Toward Agile Systems Engineering Processes. CrossTalk, April 2007.

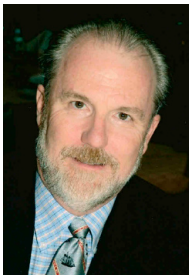
ABOUT THE AUTHORS

COL David Moore assumed command as the PM BC in August 2007. PM BC develops and sustains Army and Joint software command and control capabilities from tactical to strategic levels. The scope of PM BC includes the lifecycle management of software products that include Command Post of the Future, Advanced Field Artillery Data Systems, Global Command and Control System – Army, Battle Command Sustainment and Support System, and a common software effort. PM BC also serves as the Army's Component Program Management Office for Joint Command and Control as well as manages the Army's Common Hardware Systems procurement effort. COL Moore's previous acquisition assignments have focused on warfighting solutions for combat vehicle, soldier equipment, and software command and control systems for the joint force and coalition partners.

PEO C3T PM Battle Command
ATTN: SFAE-C3T-BC
6007 Combat Drive, 5th Floor
APG, MD 21005-1846
E-mail: PAOPEOC3T@conus.army.mil

Portia Crowe is the chief engineer and technical director for the Army, Program Executive Office C3T-PM Battle Command-Strategic. Crowe is a Ph.D. candidate at Stevens Institute of Technology, Hoboken, NJ and currently conducting research in agile systems engineering. She has received numerous Army awards for successful implementation and fielding of strategic programs, and for various research and development projects. Crowe has experience in systems and software engineering and tactical and strategic program lifecycle management.

PEO C3T PM Battle Command
ATTN: SFAE-C3T-BC-TMD
6007 Combat Drive, 5th Floor
APG, MD 21005-1846
E-mail: PAOPEOC3T@conus.army.mil



Robert Cloutier, Ph.D. is an associate professor of systems engineering in the School of Systems and Enterprises at Stevens Institute of Technology. Cloutier has more than 20 years experience in systems engineering and architecting, software engineering, and project management in both commercial and defense industries. Industry roles include lead avionics engineer, chief enterprise architect, lead software engineer, and system architect on a number of efforts and proposals. Cloutier's research interests include model-based systems engineering and systems architecting using Unified and Systems Modeling Languages, reference architectures, systems engineering patterns, and architecture management. Cloutier has a bachelor's degree from the U.S. Naval Academy, an MBA from Eastern College, and a doctorate in systems engineering from the Stevens Institute of Technology.

School of Systems and Enterprises
Stevens Institute of Technology
Hoboken, NJ 07030
Phone: (201) 216-5378
E-mail: robert.cloutier@stevens.edu

Power and Influence Charting

The Google Way

Sharon Berrett, Idaho National Laboratory

Troy Hiltbrand, Idaho National Laboratory

Abstract: The success or failure of a project may be charted in the initiation phase. Therefore, initiation is arguably the most important phase of any project. During the initiation phase, the foundation for the project is established, including the selection of project sponsors and champions and getting their buy-in, which sets the project up for success.

Introduction

The most successful projects are those that have a solid foundation and actively supportive sponsors. However, the initiation phase can be challenging because of the number of details that must be defined to ensure project success.

One key issue that must be addressed during project initiation is the identification of the "right" project sponsor; one who has enough political clout and backing to overcome the obstacles that arise in the lifecycle of any project. In large and complex organizations, this task can be overwhelming because the organization hierarchy is often dispersed geographically encompassing multiple time zones on multiple continents.

Identifying the right project sponsor is a critical step, but is difficult to accomplish. Reliance on the organizational hierarchy to identify individuals is one way, but is not always optimal.

To understand "true" power, it is important to understand what power is and how it manifests itself within an organization. According to the classic publication by French and Raven [1], there are five main types of power: legitimate, referent, expert, coercive, and reward.

Legitimate Power:

Power that is inherent to a role within the organization and not the person occupying the role.

Referent Power:

Power that comes from being liked and respected by those around you. This power is based on the fact that individuals are striving to be like you and follow your lead. It is inherent to the individual and not the role.

Expert Power:

Power that comes from others needing what you know and what you can do. It is inherent to the individual and not the role.

Coercive Power:

Power that is derived from forcing others to do that which is contrary to their own will through coercive means. This power could be based on punishment or through forceful means. Ability to execute this type of power could be based in a role or be an attribute of the individual.

Reward Power:

Power that is derived from coaxing others to do your will through promise of reward. This reward could be tangible or intangible, but the promise of a reward upon completion of the activities or set of activities is the basis for the power. Ability to execute this type of power could be based in a role or be an attribute of the individual.

In determining where this power exists within the organization, the organization hierarchy does a great job of modeling out legitimate power; however, it does not clearly identify individuals with either referent or expert power. In some circumstances, these individuals can have greater influence within the organization than those with legitimate power. Therefore, it is crucial to identify individuals with referent or expert power when determining optimal project sponsorship during the project initiation phase. Modeling expert and referent power is more challenging than modeling legitimate power, but the results are invaluable to understanding the true picture of organizational power.

To get a glimpse into how mapping the organization's power structure can occur, we can look into the history of how Google rose from an idea dreamed up in the dormitory of two grad students to one of the world's largest and most formidable companies in less than 10 years and merge that with a concept from one of the leaders in business research and analysis.

Google

In the late 1990s, there were a handful of major search engines fighting to gain market share in the search market. AltaVista, Excite, Yahoo, and several others had established themselves as internet search leaders. Indexing the World Wide Web was accomplished through a limited number of standard approaches.

The first method was to "crawl" the internet and identify all web pages that were linked together. Once a web page was found, the page content was used to rank how applicable it was to the search term the user submitted. This process was relatively simple and allowed users to find pages they were looking for. The dilemma was that just because the pages contained the search term, did not necessarily mean the intended needs of the user's

search were met. The other problem was that this approach allowed marketers to load a page with superfluous search terms to drive their web page higher in the ranking, even when there was no direct linkage between the search term and the web page.

To augment this type of indexing, these search engines also created human-managed indexes. Human reviewers would take the most sought after terms and put them into a hierarchy that could be easily searched. As they reviewed pages found in the web crawl, they were manually categorized and ranked with relative priority to other pages in that category. The outcome of these people-generated results were highly acceptable because they targeted returning content that people wanted to see and not just content that matched the search terms. It was limited by the fact that it was not highly scalable. With millions of pages having constantly changing content on the Internet, it was impossible for a person, or even a team of thousands of people, to track these pages.

This is where Google revolutionized the search industry. Although implementation was fairly complex, the concept behind Google's idea was simple. They identified that the only way to have a usable, maintainable index was to develop a way to generate meaningful search results without human intervention. Google envisioned an algorithm to automate the process of page categorization and ranking that would not rely on an individual constantly reviewing pages to keep them fresh and up to date [2].

The basic premise of this methodology was to rely not only on the content within the page, but to consider what other sites were linked to that page, the relative importance of those sites, and how many other pages that site was linked to. With the combination of these factors, Google was able to achieve meaningful results that were scalable as the Web grew. This process has been commonly referred to by Google as PageRank.

Google's definition of PageRank [3] states, "PageRank reflects our view of the importance of Web pages by considering more than 500 million variables and 2 billion terms. Pages that we believe are important pages receive a higher PageRank and are more likely to appear at the top of the search results. PageRank also considers the importance of each page that casts a vote, as votes from some pages are considered to have greater value, thus giving the linked page greater value. We have always taken a pragmatic approach to help improve search quality and create useful products, and our technology uses the collective intelligence of the Web to determine a page's importance."

One of the concepts Google has strived to continually maintain is to avoid manual intervention in the search algorithm. If issues were found in the ranking of a page, the algorithm was evaluated to identify how it could be optimized to rank that page. Google's purist philosophy has been challenging to maintain, but has also garnered trust from the user community. This confidence allows users to feel like they are getting the best results available and not the results that are best for the highest bidder.

The same innovation that propelled Google from obscurity to the top of the search industry can be applied to organizations to identify individuals who have referent and expert power, but don't necessarily show up at the top of the organizational chart. This ensures that all vital project stakeholders are identified in a quantifiable method.

Gartner Power Mapping

Gartner, a highly respected thought leader in the business research and analysis sector, has identified and published a method similar in nature to the early search engines. This method relies on knowledge of key individuals to evaluate and derive measures for an individual's power and influence in the organization. This method is called "power mapping" [4].

Power mapping is focused on smaller sized groups and its purpose is to identify which stakeholders have the most power and influence within that group. To accomplish this, the evaluator lists all stakeholders who are potential influencers. Then, the evaluator establishes categories with highest importance to the organization in terms of what power looks like in the areas of legitimate, expert, and referent (referred to as position, knowledge, and relationships by Gartner). Each individual is then evaluated on a numeric scale and the scores are added up to ascertain the overall power of each individual. The results are then vetted out through a series of interviews to ensure assumptions made in the scoring are correct. The final score represents the overall power and influence of an individual within the organization.

	Total	Position	Knowledge	Relationships
Stakeholder 1	6	3	1	2
Stakeholder 2	5	3	1	1
Stakeholder 3	7	2	3	2
Stakeholder 4	6	1	2	3

Table 1: Gartner Power Mapping

Like the early search engines, this process is extremely effective because it relies on human understanding of power throughout the organization and includes a validation process to ensure who key stakeholders are and their relative power and influence within the organization.

The manual nature of developing the power map in this fashion is very time consuming and requires institutional tacit knowledge, and changes in the organizational power base do not surface quickly. Consequently, manual development of a power map is neither scalable nor highly maintainable over the long term.

Gartner Meets Google

Here is where the concept that Google used to revolutionize the search industry can take the power map to a whole new level of scalability, maintainability, and adaptability. If the process can be automated and an algorithm developed to measure the influence and power of all individuals within an organization, then it can be scaled and updated regularly to capture power changes in the organization. In addition, the automated power and influence chart would be impervious to the need for an organizational expert's participation in the creation and maintenance of the chart, making it more resilient from a knowledge transfer perspective.

We used this concept at Idaho National Laboratory (INL) in an effort to automate the power and influence charting process and to identify the influence base within the organization. This approach allows the identification of key strategic partners throughout the laboratory who could be engaged to champion project efforts that align strategically with achieving key mission goals.

Background

In operation since 1949, INL is the Department of Energy's (DOE) lead nuclear laboratory and is dedicated to supporting DOE's missions in nuclear energy research, energy and environment, and national and homeland security. INL is operated by Battelle Energy Alliance and participates both independently and jointly with other labs in the support of work for DOE and other government agencies.

Process

The main process for mapping power in the organization follows these steps:

1. Identify intelligence sources
2. Map intelligence in categories of power
3. Gather data
4. Normalize data
5. Weight categories of influence and power
6. Summarize individual influences and power
7. Categorize individuals

Our first task was to identify which organizational artifacts would serve as intelligence sources. Key information was not available in a single consolidated system, but across the organization in the form of both structured and unstructured data. Structured data is where each data element is defined and it is possible to identify relationship between the elements, whereas unstructured data is in free form without definition or relationships.

Data had to be mined and consolidated and then classified into the areas of legitimate, expert and referent power. Often a single intelligence source was used to identify more than one type of power depending on the information extracted from it. Once this data underwent a process of classification and weighting, the relative influence that each individual has within the organization was derived and individuals were categorized making the information actionable.

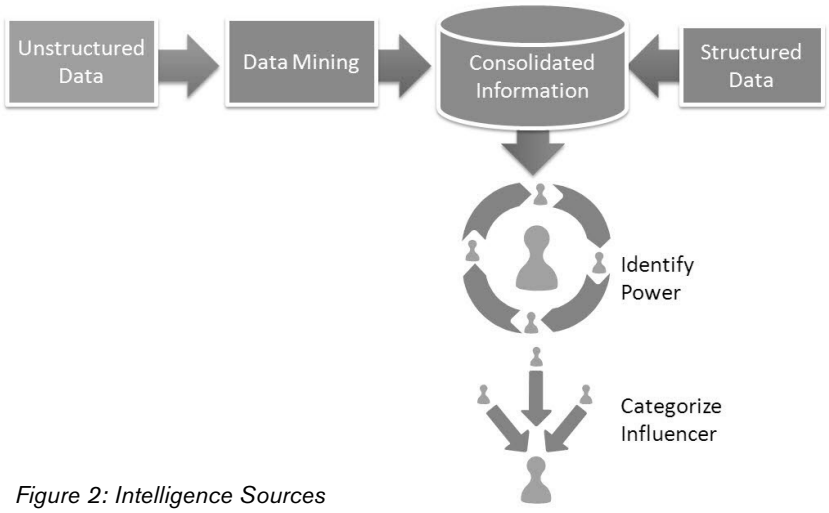


Figure 2: Intelligence Sources

Legitimate

Legitimate power was the easiest to measure. To evaluate legitimate power within an organization, we were most concerned with the span of control for that individual. Span of control addresses how many people each individual manages and who those individuals are. When identifying span of control, both direct manager-employee relationships as well as matrix manager-employee relationships were assessed. Within INL, there are two additional organizations that reflect legitimate power outside of the organization hierarchy. Councils represent the oversight of investment and management systems represent oversight of processes. Different roles within these two organizations were assessed to identify an individual's legitimate power.

Expert

With expert power, we looked at accomplishments of individuals across the organization. To identify notable individuals, the first area we examined to identify expert power was INL's internal communication system. The centralized communication system allows for notes to be distributed across the organization. These notes communicate promotions, accomplishments, upcoming meetings, areas of research, or any significant information to managers and/or employees. We gave credit to each of the individuals mentioned in communications, weighting newer communications higher than the older communications.

The second area we examined to identify expert power was key strategic projects within the laboratory. These are areas of high interest to DOE and are critical for accomplishment during the fiscal year. Each key strategic project has multiple people acting in different roles. Each of these different roles within the strategic project was given a weight as to the influence exerted over its successful completion.

Referent

Referent power deals with connections within the organization and was the most challenging to identify. Similar to the method used by Google to rank pages, organizational connections are where whom an individual knows is more important that what the individual knows.

To accomplish this evaluation, we looked at a number of existing intelligence sources used in deriving legitimate and expert power to identify the referent power. When individuals are related within these intelligence sources, it is an indication of an organizational association between these individuals. The more associations that an individual has represents the higher the likelihood that the individual has referent power in the organization. Referent power is much more than who is friends in the organization, it establishes which individuals have influence over others to make things happen. To assess this, we looked at relationships among individuals on the councils, key strategic projects, and management systems.

Figure 3: Referent Power Connections

Normalization

Since each of these factors generates results of different quantitative magnitudes, they must be normalized so that they can be combined. The goal in normalization was to take data sets with different domains and allow them to be added. Span of control might have values from 1-30, communication notes might have values from 1-5, and connections might have 1-1000. Just adding these numbers together would skew certain categories much too high in the evaluation of power.

Using some basic statistics, each number can be represented as the number of standard deviations from the mean (or the z score), putting a majority (99.9%) of the data within a normalized range and allowing it to be combined.

$$category\ score = \frac{x - \overline{x}}{\sigma}$$
$$x = individual\ score$$
$$\overline{x} = average\ for\ sample\ (excluding\ 0s)$$
$$\sigma = standard\ deviation\ for\ sample\ (excluding\ 0s)$$

Equation 1:

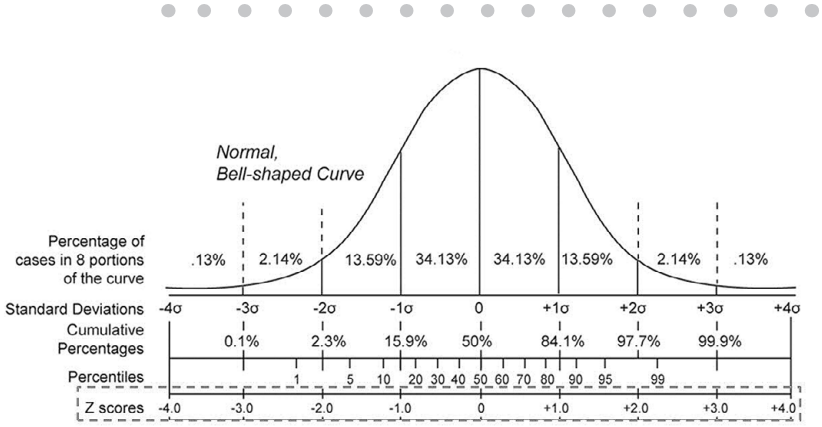


Figure 4: Normal Bell-shaped Curve

Not all data sets involved in the calculation are distributed normally and fit the standard bell shaped curve. In addition, there are often outliers in the data that have to be evaluated and addressed. In the case that data is skewed away from normalcy, other more advanced statistical methods are required to increase the relevancy of the overall power score.

We then evaluated each of the categories to identify which had the highest impact on the power for the individual. Each category was given a numeric multiplier to indicate its overall importance to the power base of the individual. These weights were then applied to each category's score and all of the scores were summed up to get a final power score.

Along with the identification of sources and calculation of factors, weighting of the categories is one of the most important aspects of this process. With multiple sources and factors participating in the overall score, a refinement of these weights is necessary to ensure result validity. To perform this refinement, the process is run iteratively, generating sets of results that can then be evaluated by knowledgeable individuals within the organization. Once their feedback is gathered as to the accuracy of the power scores, analysis is done to determine reasoning behind both false positives and false negatives and both the weights and distribution-based calculations are refined to more accurately represent the nature of power in the organization.

This power factor describes the relative level of influence and power an individual has in the organization. Since connections within the organization are reliant not only on how many connections an individual has, but also the relative influence score that those connections have, it is important to run the calculations through a number of times. The first time, all individuals in the organization have an equivalent influence factor. Each successive time that the calculation runs, the new power factors are used and the relative power factor exerted on the connection get closer to representing the truth. Each successive time the calculations are done, the influence factor changes some, but as it is done multiple times, that change gets smaller and smaller until it approaches zero. This gives us the most accurate representation of an individual's power score within the organization.

Categorization

This power score is a relative representation of the influence of the individual in the organization, but unless there is assurance that all intelligence sources were utilized and the weights are accurate, it can be misrepresentative of the exact influence of an individual. To simplify the usage and establish more usability to the number, we broke these into categories of influencers associating levels of influence based on their relative scores.

In Practice

At INL, this process has been instrumental in helping to identify influential stakeholders. In mid 2010, Information Management (IM) was given the charge to lead up efforts to transform the workplace at the laboratory through an initiative called High Performance Workplace. Since this initiative involved culture, information and process and not simply a technological change, it was imperative to identify influential stakeholders throughout the laboratory that would act as change agents for the initiative. Through use of the power map, we compiled a list of individuals throughout the organization with whom we could engage to

generate the "grass roots" support of the initiative to execute effective change. This distributed engagement with key individuals both at the management level and the organization level allowed us to ensure both a top-down and bottom-up approach to organizational change management. This approach has established a framework for success for the initiative.

Conclusion

With this categorization of employees, we have the capability to have a better understanding of where the true power in the organization lies. It also helps us to determine key individuals in the organization, which serves as one input into the decision making process for project initiation based on the relative importance of the request to the organization.

Through the application of methods and innovation that propelled Google to the top to a strategic toolset from Gartner, we were able to create a sustainable and objective manner to facilitate in the project initiation phase.

Disclaimer:

STI Number: INL/JOU-10-19794

REFERENCES

1. French, J. &. 'The bases of social power,' in D. Cartwright (ed.) Studies in Social Power. Ann Arbor, MI: University of Michigan Press. 1959.
2. Stross, R.. Planet Google. New York City, NY: Free Press. 2009.
3. Google, I.. Technology Overview. Retrieved April 29, 2010, from Google: Corporate Info: <http://www.google.com/corporate/tech.html> 2010.
4. Mesaglio, Mary; Aron, Dave;. Leading from your Power Base: Toolkit. Gartner. 2007.

ABOUT THE AUTHORS

Sharon Berrett has been employed at INL for 31 years. For most of that time, she worked a project manager overseeing various IT and research projects. She worked in the IM Strategic Planning organization as a strategic liaison. Her most recent assignment is as a strategist in the Portfolio Management Directorate. Ms. Berrett is PMP certified and holds a Bachelor's of Science in Business Administration.

Sharon Berrett
Idaho National Laboratory
P.O. Box 1625
Idaho Falls, ID 83404
E-mail: Sharon.Berrett@inl.gov
Phone: 208-526-9629

Troy Hiltbrand is the manager of IM Strategic Planning and Enterprise Architect at INL. In this capacity, he is involved with coordinating efforts to align IM activities with Laboratory strategy and vision. Mr. Hiltbrand is PMP certified and received a BA from Utah State University in 2000 and an MBA from Idaho State University.

Troy Hiltbrand
Idaho National Laboratory
P.O. Box 1625
Idaho Falls, ID 83404
E-mail: Troy.Hiltbrand@inl.gov
Phone: 208-526-1092

Product Thinking in Process Improvement

Terry Leip, Intel Corporation

Abstract. We are often told that process improvement activities should be managed as a project, but seldom do we hear that they should also be managed as a product. Key decisions ranging from high level strategies to the deployment of improvements can become much simpler when we view the approach of our process improvement work in the same way we would for the development of more conventional software products. This article discusses six examples of this concept that will help you not only simplify process improvement decisions, but improve the odds of success in your process improvement activities.

Background

More than six years ago, Intel Corporation's 5,000-person IT organization embarked on a journey to improve its internal development processes in an effort to increase development efficiency and address customer satisfaction issues. One of the key approaches learned by the author during this time was to shift his thinking from "Building Processes" to "Building a Product;" in essence, to apply the lessons learned in his software development to process improvement activities.

1. Know Your Market

Software products typically have marketing plans that identify target markets, the size and composition of the segments within those markets, and a description of the customers within those segments and their needs. These plans allow products to be focused on meeting the most important needs of the most valuable markets. In addition, this enables products to be created that are matched to the skill and experience level of users in those markets.

Our early attempts at process improvement work were lacking this type of information and as a result, our process improvement team assumed our market was large, high-risk, and long-duration projects that were led by very experienced project managers. In reality, the majority of the projects were smaller, shorter, and lower risk than had been assumed, and many were led by individuals who were inexperienced in the role (Figure 1).

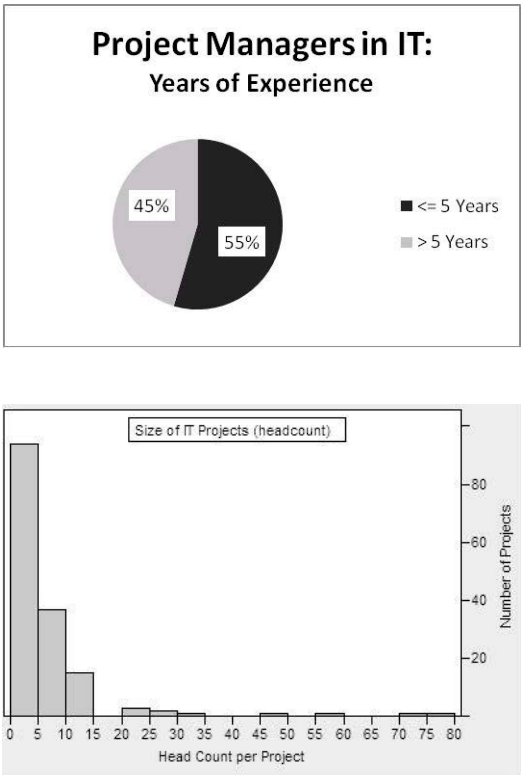


Figure 1: Sample aspects of target market (projects in IT)

The inevitable result was that the process improvements did not meet the needs of the user base; both informal feedback and audit data showed that processes were neither well received nor widely used.

Through this and other similar experiences, we learned that we could not make assumptions about our customer base and expect to be successful. Even though our understanding of our markets continues to develop, we now know a great deal more about how the typical project looks in terms of size, experience, duration and other key attributes. The most recent releases of processes and training have been rewritten with these characteristics in mind so that we are not only better meeting the needs of the projects, but we are also no longer supporting

material which isn't being used. In a "before and after" survey from our users (Figure 2), we found that customers feel that our product is much easier to use as well as more applicable to the size and complexity of their projects.

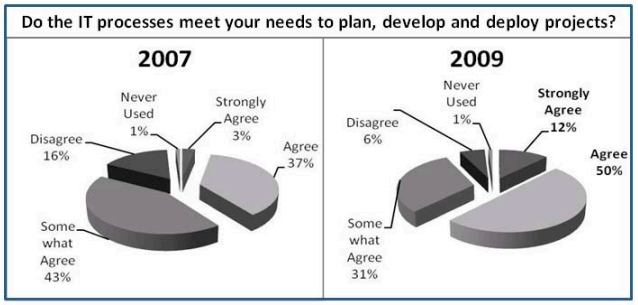


Figure 2: 2007 vs. 2009 Project Manager Survey Results

2. Product Architecture is Important

In software development, a good architecture helps ensure that a product not only meets the requirements, but that the product can be more easily maintained and extended over time.

Four years of documentation changes combined with the lack of defined process architecture left us in the situation where process resided in templates, training had found its way into in our processes, and policy had become intermixed with processes. The resulting patchwork of documents had become extremely time consuming for us to maintain and difficult for users to find needed information quickly. In addition, this lack of structure also resulted in users being required to enter the same information in multiple locations.

Faced with increasing maintenance effort and a chorus of customer feedback, we performed an evaluation of the required data for all of our templates and tools. We attempted to identify where the same information was required to be entered in more than one location. This simple analysis yielded some surprising results: For a cascading waterfall lifecycle project (about 70% of our projects), there were a total of 150 duplications of required data over the life of the project. The single largest offender was the problem statement, which was duplicated in a total of six different locations (Figure 3).

In addition to the reductions of required data, we defined a process architecture that included a clear definition of what type of information resided in which type of documents. We then kicked off a project to rewrite our process materials following that architecture, as well as targeting the aforementioned duplications. The results from this effort were very rewarding (a reduction of redundant data entry by more than 70% and positive survey feedback from our customers), however the rework has been costly and could have likely been avoided if an architecture had been clearly defined before our processes were initially developed.

3. Beware of "Free Features"

When building software products, it has been often said that there are no free features; everything must be developed, tested, supported, and maintained. In software there is often the temptation to toss in a few seemingly simple features or to add a minor last-minute request from a key customer. The problem,

D=Data S=Status L=Link	TP-20 Project Ma	Plan	Value Decision	Comm
Requirements ID (High Level)				
Roles & Responsibilities	D, S			
Requirements, High Level		D		
NPV / PI		D	D	
CMMI Scorecard	S	D	D	
Assumptions		D	D	
PLC Approval Decision		D	D	
PLC Meeting Agenda		D	D	
PLC Meeting Objective		D	D	
Strategies		D	D	

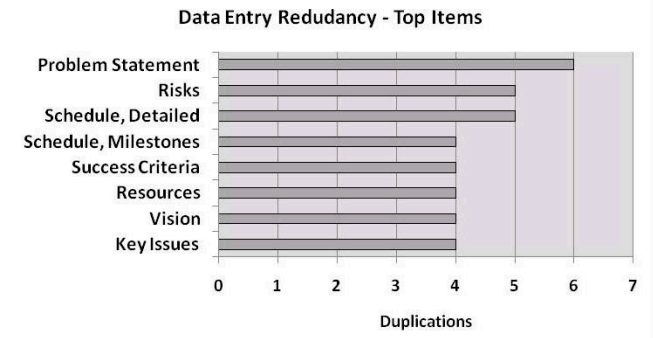


Figure 3: Sample of data entry redundancy analysis

of course, is that maintenance activities are often 60-80% of the total costs of software and every one of these free features must be maintained regardless of their value to the customer.

We have found the same is true with our product; every extra process step, guideline, template, or checklist requires development, testing (i.e., reviews), support (training, coaching, and auditing), and maintenance. We had fallen into a habit of saying "yes" to almost every customer request in an attempt to please our customers, regardless of the scope of use or cost to maintain. As a result, our suite of documentation had grown to more than 1,400 pages spread between 200 or more documents, and even minor changes required updates to a multitude of areas: Process, templates, guidelines, checklists, examples, training, etc. At one point, we had two types of requirements templates and three types of test plan templates with detailed examples to accompany each one. Based on often isolated requests, we had created detailed process steps for seldom used processes and built templates which included far more data items than the majority of the users ever needed.

Our 2008 process release focused on removing these free features that had crept into our product. We condensed the number of similar templates, dropped the examples, and took a bold step to reduce our process documentation and templates to the absolute bare minimum possible (typically one to two pages per document, just focusing on the essentials).

The results were dramatic; we shrank our collection of processes, templates and checklists by more than half (Figure 4), and we are projecting savings of more than $25,000 per year in reduced internal maintenance costs, and users actually found that the resulting materials were easier to use.

Collateral	Pages	Lines	Words	Files
Before	1,405	45,970	389,136	213
After	636	20,814	177,505	144
Reduction	55%	55%	54%	32%

Figure 4: Reduction of documentation size

We also recognized that we needed to address this stream of low-value features and change our current behavior to prevent our product from winding up with the same problems in the future. To do this, we beefed up our process for handling feature requests, including creating a simple tool for scoring change requests as high, medium, or low against four value drivers (Figures 5 and 6).

Criteria	
Value (user)	Low
Scope of Use	Low
Cost (user)	Low
Cost (PMO)	Med
Score:	2.8

Figure 5: Feature request 'scoring tool'

Feature Request Score Legend		
Overall Rank	**Description**	**Score**
Red	**Poor candidate** Not recommended to implement	< 4
Yellow	**Possible candidate** A 'nice to have' item, ZBB candidate when resource constrained	4 – 10
Green	**Good candidate** Recommend to be candidate for implementation	> 10

Figure 6: Feature request scoring legend

This tool produces an overall score that gives us a consistent standard for ranking feature requests and supports rejecting those requests that do not have sufficient value to justify the costs involved to our customers and our process development team.

Using this approach and tool, we have been able to reduce the number of low value changes (our overall rejection rate went from 10% to 33%) and most surprisingly, users have been very understanding when we say "no," in large part due to being made aware that a standard set of criteria is being used in making the decision.

4. Change your User Interface Cautiously and Infrequently

In software projects, development teams are careful about radically changing the "look and feel" of an established application because they know that once users are accustomed to using their product, they will need to spend valuable time to relearn things that have changed. In process improvement, the interfaces are the templates, process documents, and web sites where they are contained.

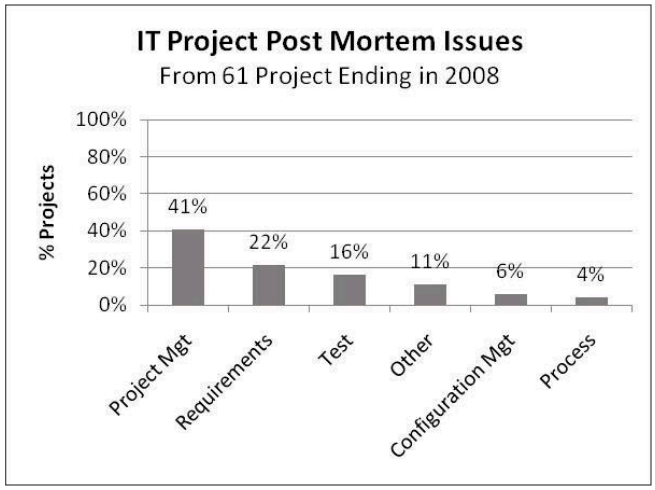
We learned this lesson when we performed our first round of process improvements in 2005 [1] to reduce the complexity of key areas of our processes and templates. When the updated materials were deployed in the organization, we were surprised to discover that many users were upset by our improvements because they had grown accustomed to the look, feel, and the steps of the previous processes. Even though the new versions had fewer and simpler steps, they still required the project teams to spend the time and effort to learn the new interface.

The key learning was that our users want to spend their time and energy doing their work rather than relearning new interfaces to our product. In the intervening years, we have become much more cautious about changing interfaces and have learned a great deal regarding how to minimize the impact of changes. One major improvement was to simply reduce the frequency of large changes and we now work on a cadence of two releases per year, and any substantial changes to interfaces that must happen are grouped into one of those releases rather than dribbling them out over time. This more predictable approach has greatly reduced the complaints and anxiety that we experience with past changes.

5. You Cannot Just Ask Users What They Want

One of the most commonly encountered issues when discussing software requirements is that you can not simply ask users what they want and expect their answers to be correct and complete. It is not that users do not want to provide good requirements, but that they are often too close to the issues and often have many unstated assumptions about how they really work. People developing software requirements know that multiple elicitation techniques should be used to properly understand the real problems to be solved and uncover the true requirements.

In the early days of our process work, we routinely collected our process improvement feedback from customers during quality assurance audits and via change requests from any users that cared to submit them. The bulk of these suggestions were for "less process" or minor changes to templates or processes, but seldom addressed issues that would improve overall project execution. Last year, we analyzed the results from more than 60 project postmortems and discovered that virtually none of the issues or solutions identified by the postmortems related directly to our user feedback (see Figure 7).



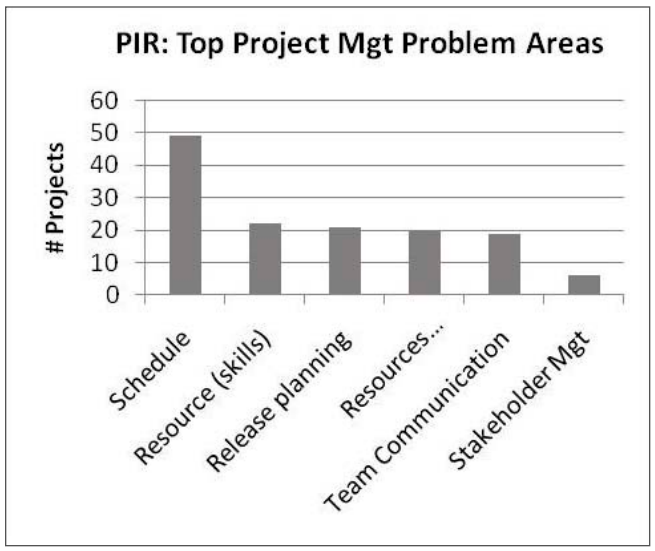
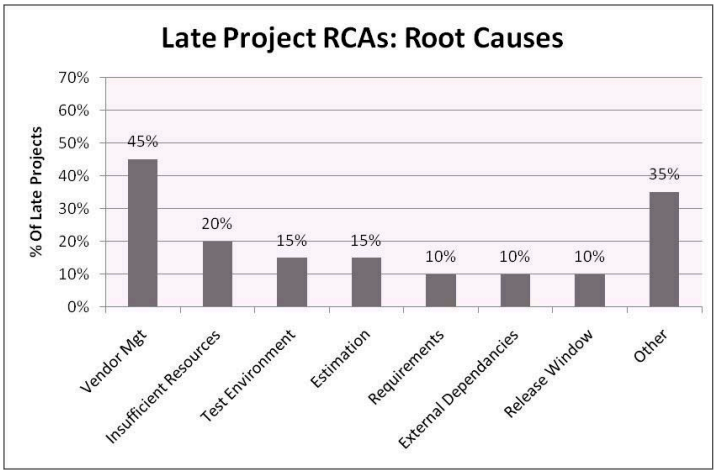


Figure 7: Roll-up of Project Post-Mortem Issues



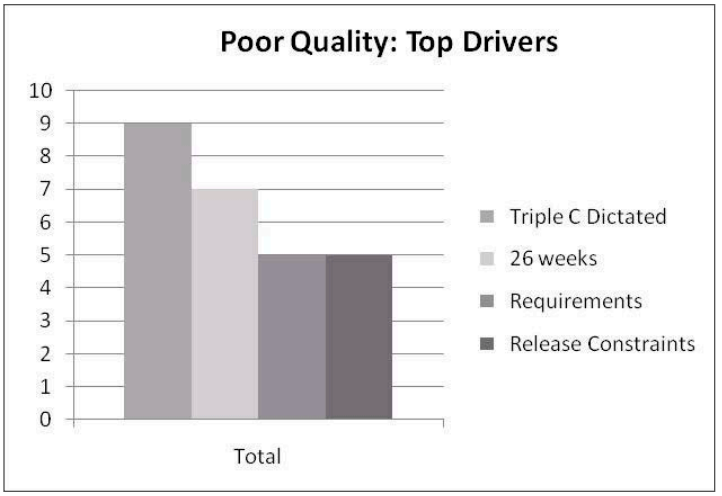


Figure 8: Samples of customer issue data

Given that one of our key business objectives was to meet delivery commitments, we recently started performing formal root cause analysis of projects that missed their committed release date. This data has shown that issues with the performance of vendors and sufficient resources are the drivers for 65% of late project deliveries. In addition, we recently engaged in more sophisticated business-problem focused interviews with randomly selected project managers and test leads to uncover issues impacting key outcomes such as product quality. The results of this activity identified issues such as management dictating scope, schedule and resources, and an IT mandated 26-week project duration limit as large drivers of quality issues (see Figure 8).

While we are still struggling to gather our requirements using more sophisticated methods, our understanding of our customer requirements has improved substantially since we have moved beyond simply asking our customer, "How can we improve our processes?"

6. Provide Strong Customer Support

Many of us have the experience of calling a customer support hotline for a product and after navigating the phone menu waiting for an extended period of time. When we finally speak with a real person, we then discover that they have insufficient knowledge to help us. The key is that without solid support, many customers may become frustrated and simply give up on a product and not bother to use it.

One of our earlier and better decisions was to provide process coaches and quality assurance auditors who not only knew the processes, but also had real-world experience using them on projects. This allowed them to help our customers to understand both the value of the processes and how to appropriately apply them to projects. This support was instrumental in ensuring that project teams did not give up in frustration when they did not know how to use a process or tool. Feedback from project teams has been overwhelmingly positive towards this effort, with many project teams indicating that they would not have been able to adopt the processes without the support. The bottom line is that if it is too hard to use a product, customers will not use it fully or will not use it at all; the same is true for any process improvement "product" as well.

Conclusion

I have provided only a few examples of applying "product thinking" to process improvement, but I continue to find new applications nearly every day. The key is to step back from decisions and ask the question, "How would we address this process improvement issue if this was a software product?" I think you will discover that many formerly perplexing process improvement issues become clearer and solutions become obvious when you make this key shift in your thinking and approach.

Disclaimer:

Other names and brands may be claimed as the property of others Intel, the Intel logo, Intel. leap ahead., and Intel. Leap ahead. logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

REFERENCES

1. Brodnik, Mark, et al. "Why Do I Need All That Process? I'm Only a Small Project", CrossTalk February 2008

ABOUT THE AUTHOR

Terry Leip is a Process Engineer in the Operational Excellence group at Intel Corporation's IT Flex Services. He has more than 20 years of software development and quality experience with Intel and Lockheed Martin. He holds a bachelor of science degree from Grand Canyon University, is a Six Sigma Green belt and an SEI authorized CMMI SCAMPI B/C Team Lead.

E-mail: Terry.Leip@Intel.com

Software Estimation

Developing an Accurate, Reliable Method

Bob Sinclair, NAWCWD

Chris Rickets, NAWCWD

Brad Hodgins, NAWCWD

Abstract: From a management perspective, it is essential that software estimates used in a TSP launch are as accurate as possible. Significant growth due to estimation inaccuracy can wreak havoc on a team attempting to stay within cost and schedule while executing its established plan. This article discusses how a software team that uses both proxy-based and size-based estimates is able to accurately plan, launch, and execute on schedule.

Introduction

The AV-8B Software Development Task Team has successfully maintained and enhanced avionics and support products for the Harrier II aircraft for the better part of a decade. While there are several factors that contribute to its success, a key element is the team's ability to provide timely and accurate cost and schedule estimates to its management and customer. This was not always the case. When the team first began preparing software estimates, it was ad-hoc. At that point, neither the Software Development Task Team nor its management had faith in the estimates. When the team adopted the Team Software Process℠ (TSP)/Personal Software Process℠ (PSP), it became a priority to define and document accurate estimates. In order for a team to execute a successful TSP/PSP project, the tasking estimates need to be well defined and communicated. If not done, the team will not buy into the resulting schedule and plan which could put the project in jeopardy of failure.

Background

The Naval Air Warfare Center Weapons Division (NAWCWD) AV-8B Joint System Support Activity has successfully applied TSP/PSP for software development and maintenance projects for nine years.

This began in the spring of 2002 when the software development task team began the H2.0 block upgrade maintenance software effort [1]. Since then the software team has completed an additional four block development efforts (H4.0, H5.0, H5.1, and H6.0) and is currently working the H6.1 block development effort. The block efforts typically last approximately two years and incorporate the TSP/PSP framework.

Up until 2002, all estimates were performed by a single individual, the lead software engineer. These estimates did not follow a documented process, much less a proven method. The estimates were rough and relied on engineering judgment (i.e., the estimates were prepared using the old "thumb to the wind" method). It was up to the team to develop a consistent estimation process. However, several questions needed to be addressed as part of this effort: How would the team determine the accuracy of its estimation approach? How would they know if the estimate was complete? Would something be missed? Could a reliable schedule that the team could execute against be produced from a set of detailed estimates? To compound matters, the team found out that the program office required multiple types of estimates. These estimates were needed to support the team's management in making budgeting, planning, and build decisions.

Types of Estimates

TSP projects are initiated by a project launch. This is a four- or five-day workshop where the TSP project team develops the project plan. Key roles, goals, objectives, requirements, and constraints are established during this workshop. Most importantly, for this discussion, the team establishes a detailed estimate and an overall project schedule [2]. Therefore, the software team's launch success was predicated on the team's ability to have an accurate reliable method of performing estimations for which to generate a realistic schedule. In order to be successful, the team needed to provide as accurate of an estimate as possible, but estimates were being provided by the team having had no prior experience in software estimation and with limited resources. In order to accommodate the types of estimates needed by both management and the software team launch, the team established the following: High Level (30,000-foot) Estimate, Low Level (10,000-foot) Estimate, and detailed estimate.

High Level Estimate: This estimate is also referred to as the 30,000-foot estimate or a rough order of magnitude and does not contain details since, at that height, you would not see any details. From a conceptual point of view, management may want to integrate some new capability into the software and needs a not-to-exceed cost estimate. Typically, Technical Interchange Meetings are held for the purpose of discussing both a proposed capability (including modification to an existing one) and the general idea of how the new software would function. However,

at this level there are no formal requirements; hence, the concept of a 30,000-foot estimate. Why? From 30,000 feet, there is not enough detail to get a clear enough picture of all the areas of code that are affected or needed. This type of estimation is used in order to determine if it is feasible and cost effective to proceed into development. Once the TIMs have occurred, an estimate of this type typically takes a day or two to develop.

Low Level Estimate: Once management has determined that the new or modified capability is worth funding and appears to be within the budget, they may request a more refined (i.e., more accurate) assessment. Additionally, the functionality of the proposed capability may be reduced or increased, depending on the budget available. At this point, there is typically a better understanding of what needs to be done. Level 1 requirements (high level system requirements) may be available, along with view graphs calling out detailed functionality; hence, the concept of a 10,000-foot estimate. Things are a little clearer and better defined.

Detailed Estimate: The detailed estimate is performed prior to, and in preparation for, the TSP Launch. During this phase, the Software Engineer (SWE) that is preparing the estimate works with a Systems Engineer[1] to understand both system and software functionality and to evaluate the requirements. The SWE develops a conceptual design that identifies the initial architectural components. These components are then mapped to development tasks, which are workable sized tasks that are identified as development or maintenance tasks. The information associated with each task is documented in a standardized spreadsheet. A set of spreadsheets will be used to document the estimates for each capability with one spreadsheet per affected subsystem. At this point in the Software Estimation, Level 2 system requirements may be available, as well as data from formal program reviews. This is typically in the form of Critical Design Review or Preliminary Design Review slides and action items. These inputs are taken into consideration, if available. The tasks are then divided among the team so that they may prepare detailed task estimates. These task estimates will be documented in the spreadsheets.

Software Estimating (Proxy-based vs. Size-based)

Early on in H2.0 block development, the team realized that the lifecycle for new software development did not address problems associated with software maintenance. Therefore, a lifecycle for maintenance was developed that did not use size-based estimates but used proxy-based estimates instead. The primary reason for both the new lifecycle and the focus on proxy-based estimates is that the development pattern that is followed for maintenance is not consistent with that for new development. For example, in some cases a significant amount of time must be spent identifying the source of the problem with little time up front spent identifying the fix, followed by a significant amount of time spent verifying and testing the fix. Therefore, the software team decided to use the PSP concept of proxy-based estimation. The proxy sizes and times were adjusted over time based on actual data until it stabilized. It took approximately three years before the team identified the four proxy (size to effort) categories [3]:

PROXY	Estimated Effort
Small	6 hours
Medium	17 hours
Large	35 hours
Extra Large	60 hours

Figure 1: Proxy Size-Estimating Table

These proxy sizes have stood the test of time and have not deviated since the H4.0 block build. Originally, the software team used size-based estimates for all new development efforts and proxy-based estimates for maintenance efforts. But this was later abandoned when the team realized that both types of estimation techniques could be used with either new development or maintenance efforts. Analysis of the team using proxy-based estimates showed that the software team was accurate when estimating small and medium tasks, but the complicated, larger maintenance tasks were more difficult to accurately estimate the level of effort involved [3]. The team has developed two strategies for handling these more complicated tasks: (1) change the estimation type to size-based or (2) break the task up into small- and medium-sized tasks and use the proxy-based method on the resulting tasks.

Software Estimating Tool

In an attempt to improve estimation accuracy for large and extra large tasks, the software team developed an estimation tool to assist developers in making proxy-based estimations. At first, the software team felt that this tool was a good concept, but after using the tool for several years, the software team found more disadvantages than advantages. The advantage was that the tool provided new SWEs with a means to ensure that they did not underestimate the size of a task. The disadvantages were experienced by the seasoned SWEs. Once seasoned engineers enter their data, they would often find their engineering judgment disagreeing with the tool. When this was the case, they would simply change the answers to the questions until the tool produced what they felt was the proper proxy size or ignore the proxy size that the tool provided altogether and submit their own. Another disadvantage was that it became difficult to identify what the correct questions for the tool to ask should be, along with the correct computations and weightings to represent each question's impact on the estimated proxy size, to get around the previous disadvantage. This last disadvantage resulted in one SWE spending a considerable portion of time working on refining the size estimation tool rather than working on actual software tasking (i.e., modifying the size estimation tool had become a time-consuming, never-ending chore)[3]. For this reason, the team abandoned the tool concept and adopted establishing an estimating process and spreadsheets to capture the estimates.

A Detailed Estimating Tool is Born

As mentioned earlier, the software team needed to establish a stable way of performing estimations. Initially, estimates were captured in a text file, but this became hard to track and each estimation file did not resemble the next. The team then adopted a spreadsheet approach. At first, the spreadsheet files were simple, but over time they have evolved into MACRO-driven and organized sheets that are very effective in capturing all tasking, size, and lifecycle model information needed for a TSP Launch. This was a departure from the software estimating tool where the SWE would answer questions and the tool would factor in criteria to determine the estimation. The detailed estimating spreadsheets and their usage is described below:

Rollup Sheet: The first sheet in the file is a rollup of all other sheets that contains each component or task and its associated data (i.e., Source Lines Of Code (SLOC), lifecycle used, sub-component name, etc). All SLOC on the first sheet is rolled up at the top of the page to allow size determination. During a launch, there is typically no need to go further in the file than the rollup sheet for populating the Work Breakdown Structure (WBS) size data. A typical estimation workbook will contain the following:

Assumptions Sheet: This sheet captures any assumptions that are being made which could affect the level of effort needed to complete the tasks within the sheets.

Architecture Sheet: This sheet is used to capture the conceptual design/architecture that the tasking sheets support. Any change to design could cause tasking sheets to be added, modified or removed.

Tasking Sheets: Each component/tasking sheet contains the requirement, the files affected, description of changes to the file and the SLOC count. The SLOC is rolled up and displayed at the top of the sheet. Once all requirements are entered, files, changes, and SLOC are identified, the SWE can then determine and select from a drop-down menu the lifecycle model to be used for this component/task. If the lifecycle supports proxies, then the proxy size is also selected from another drop-down menu. Once all tasking is identified for the sheet, it can be integrated into the rollup sheet.

Once the launch is complete, these tasking sheets contain the detailed effort needed to complete each task and can now be used by the assigned SWE in determining what the assigned tasks in the WBS entail.

The detailed estimate spreadsheets works so well that it is now also used for High Level and Low Level estimating, although very little detail is added on the tasking sheets in these estimates.

Quality

The next step in the estimation process is for the estimates to be inspected. For 30,000-foot and 10,000-foot estimates, the software development task team Lead and software subsystem technical expert will review the estimate. For detailed estimates created before a launch and during the development cycle, the software team will review them as part of the final check. During these reviews, all defects including both substantive and minor documentation issues are addressed. All identified defects are reworked as required.

Estimation Currency

As mentioned previously, each capability that is produced in a block development undergoes several iterations of estimates. Initially, in order to support the customer's build decision the software team will create a 30,000-foot estimate. Later, when the customer has made the decision to build the capability, the software team will create a 10,000-foot estimate to support the customer's budgeting and funding activities. These course estimates may be updated as required by the customer. Then, before the first launch to support block development, the software team will create a detailed estimate. This estimate will support the launch activities and will result in a schedule and cost that management and the software team will work with going forward. The software team uses Process Dashboard[2] to track the development effort. It is this detailed estimate from the launch that will be used as the plan of record in Process Dashboard.

As the development proceeds, new system and software requirements will be added to the project that will require the plan to be modified. The estimates that are associated with these new requirements will be updated, as will the plan of record in Process Dashboard. Also, every six months the team revisits its capability estimates and re-launches the project. This is primarily a realignment of the team's plans to accommodate project progress and changes to the organization's direction and priorities [2]. In order to realign the project plans to the new guidelines, the team must make adjustments for requirements growth and also accommodate the addition and removal of capabilities. The result is that management has current information on the plans for completing the current block. Because the team is continually updating the task completion information in Process Dashboard, management has good quality information on the performance of the team against the plan.

Proof is in the Numbers

So how successful is this approach? Peter Russo, general manager for Microsoft's IT application architecture group comments that:

"There are two fundamental issues in most IT organizations today, one being the ability to accurately predict a project schedule, and the other being the quality of the product once you are finally done" [4].

As Russo points out, identifying a realistic and reliable schedule is essential. This, of course, cannot be done unless you have solid tasking estimates from which to create it. In addition, what is the point of meeting a schedule if the quality of the product is poor? These issues transcend the boundaries of just an IT organization and apply to any organization developing software on a timeline within a fixed budget. Figure 2 shows the actual size in SLOC of the effort for blocks H4.0 – H6.0. Note that the source size grew 46K between builds H4.0 and H5.0 and 35K between H5.0 and H6.0. SLOC size is determined by the number of SLOC that are added and modified to the existing baseline.

Block	Size (SLOC)
H4.0	47,997
H5.0	94,682
H6.0	129,607

Figure 2: Actual Size by block

Product Quality: There are different measures to indicate the quality in a product. The book, *Code Complete,* indicates that the Industry Average defect density is between 15 and 50 defects per 1,000 Lines of Code (KLOC). Microsoft applications are produced with a defect density of about 0.5 defects per KLOC in released code. Organizations using the Harlan Mills pioneered "cleanroom development" technique have been able to achieve densities as low as 0.1 defects per KLOC [5]. The software team uses defect density (defects per KLOC) to determine the quality of its products. Figure 3 compares the defect density of each block delivery against defects delivered by CMM® level 1 and level 5 organizations. Defects identified here for CMM level 1 and level 5 are captured from Capers Jones who has identified software delivered defects at each level of the SEI CMM [6]. As can be seen, the defect density for all blocks is significantly lower than that expected of a CMM level 5 organization. In addition, the quality is better than Microsoft's threshold and approaching that expected by those using the cleanroom development technique.

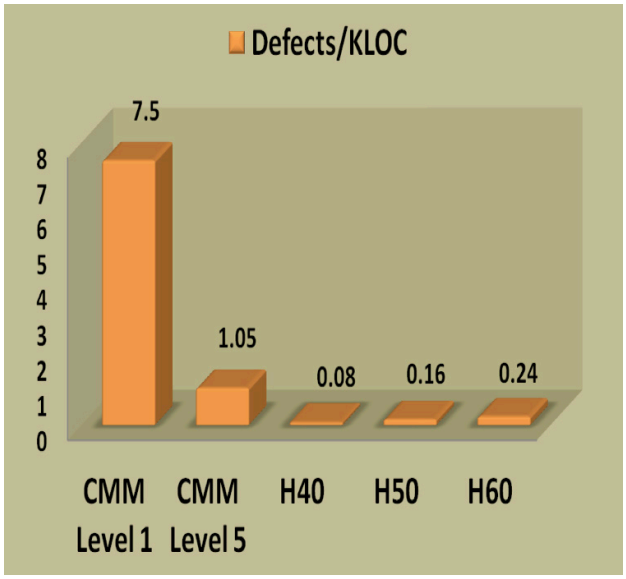


Figure 3: Defect Density

decreases as the size of the product increases. For these three data points the relationship is almost linear; the defect density increases by about 0.002 per KLOC. Other factors including task complexity and team volatility may have an affect on the quality, but were not factored into the data. That being said, the quality of the software at release is high.

Proxy Estimating Accuracy: As mentioned earlier, establishing a reliable schedule requires accurate software estimates. Given that the team is developing high quality products, figures 4-7 illustrate how well the team did at estimating task sizes indicated in the Proxy Size Estimating Table (Figure 1).

For small software development tasks (Figure 4), the software team did an excellent job identifying them and improved its estimation accuracy with each consecutive development effort.

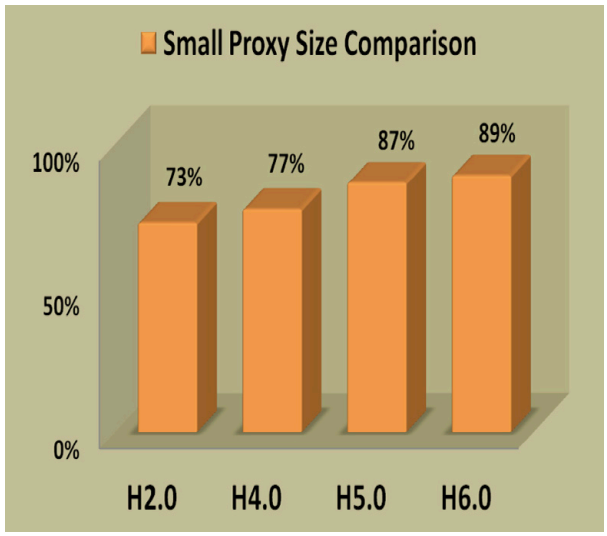


Figure 4: Small Proxy Estimation Accuracy

The number of tasks identified for each block was 11, 62, 45, and 204 respectfully. Although the number of tasks grew significantly by H6.0, the team was still able to accurately estimate this size category.

For medium software development efforts (Figure 5), the software team did a good job of identifying these tasks. They improved with each development effort given that H5.0 and H6.0 only varied by 2%. The number of tasks identified for each block

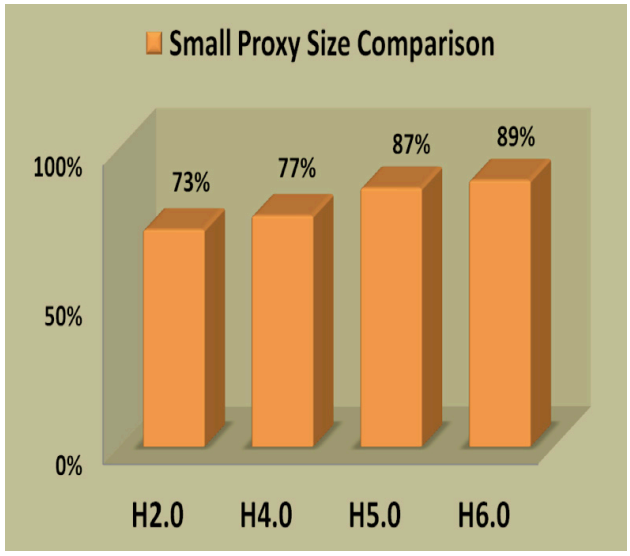


Figure 5: Medium Proxy Estimation Accuracy

was 61, 65, 50, and 291 respectfully. Although the number of tasks grew significantly for H6.0, the team was still reasonably accurate in estimating this size category.

For large software development efforts (Figure 6), the number of tasks identified for each block was 37, 29, and 76 respectfully. H5.0 did not have enough data points in this proxy category for comparison. Here the team did a good job of estimating and is improving in this area. But because tasks of this size tend to

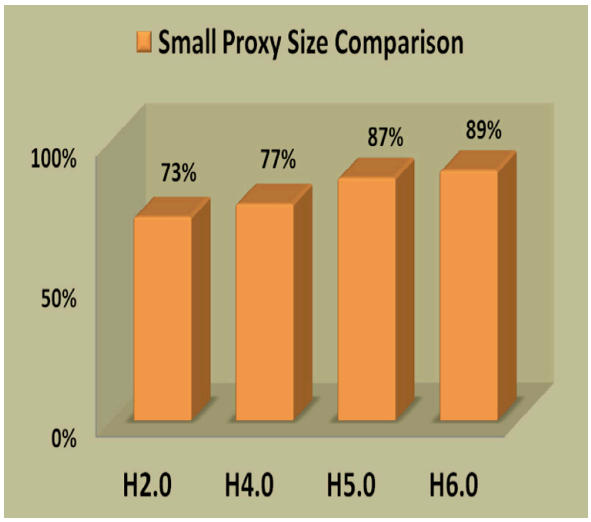


Figure 6: Large Proxy Estimation Accuracy

be more complex, it is more difficult to estimate as accurately as compared to smaller task sizes.

For very large software development efforts, the number of tasks identified for H2.0 was 18 and H6.0 was 21. Both H4.0 and H5.0 did not have enough data points in this proxy category for comparison. As one would expect, tasks this size are significantly more complex and difficult to estimate. In some cases,

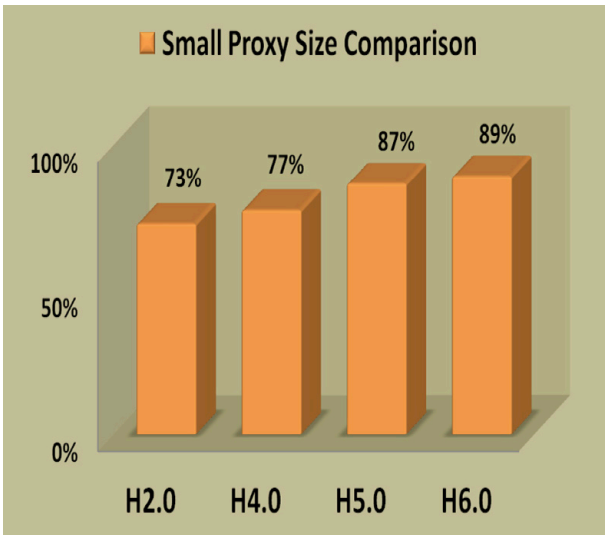


Figure 7: Very Large Proxy Estimation Accuracy

these tasks can be more of what is referred to as "science projects," where the task is known to be very complicated and has too many unknowns to determine what is required. The team has gotten better at breaking down complex tasks into multiple smaller tasks. Overall, the team trend appears to be getting better at identifying tasks of this size.

In summary, for proxy-based estimations, the software team did an excellent job estimating the number of small tasks, but as the data indicates, as the tasks became larger and more complex it became more difficult to estimate the level of effort involved. So how well does this estimation methodology support the production of an accurate plan?

Plan Accuracy: Progress against the plan is described in terms of earned value, which is based upon the estimated labor hours needed to complete each task. As the team completes tasks, they are able to determine how well they have done in meeting the plan. Figures 8-11 show how well the team's execution (earned value) compared to the plan (planned value) for blocks H2.0-H6.0.

Figure 8 illustrates the planned versus actual earned value for the H2.0 block project. Initially the actual earned value was accrued at a significantly higher rate than the planned earned

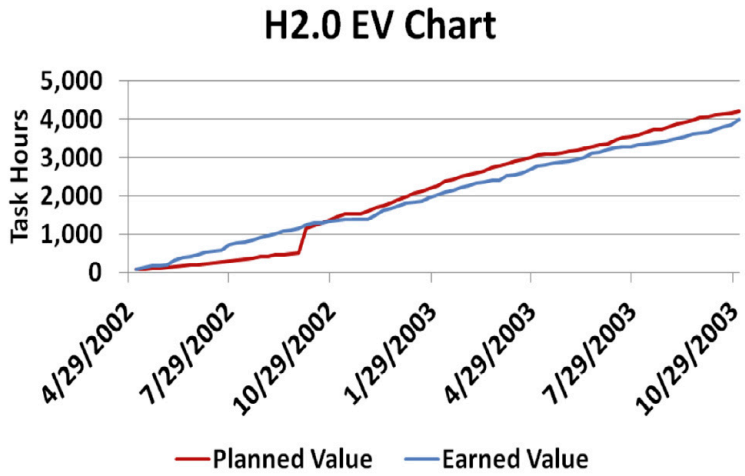


Figure 8: H2.0 Schedule Accuracy

value. This was a result of the team overestimating the H2.0 tasking efforts. At this point, the team had not yet established a reliable estimating method. A relaunch occurred where the graph of the planned value abruptly joins the actual earned value curve (October 2002). After this relaunch, the team accrued earned value more closely to the planned earned value.

The planned versus actual earned value for the H4.0 block project is shown in Figure 9. Between August and November of 2004, the graph is flat due to missing project data. The team at this point had established the estimating sheets but still had not bridged

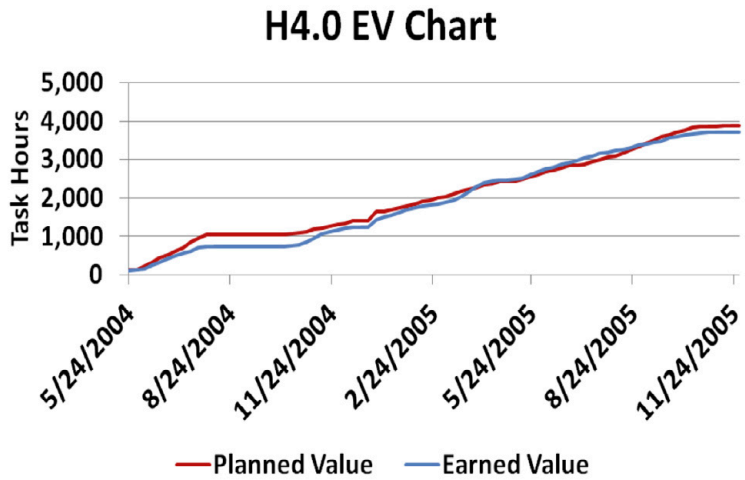


Figure 9: H4.0 Schedule Accuracy

the gap between low level and detailed estimates. Although the software team's accrued earned value followed the planned earned value relatively closely, there are numerous steep and shallow slopes of the earned value line, reflecting periods during which the team received extra credit for completing over-estimated tasks, or too little credit for completing under-estimated tasks.

By the time the software team had launched H5.0, the estimating method was fully established. Figure 10 illustrates the planned versus actual earned value for the H5.0 block project. The separation of planned versus actual earned value in the latter

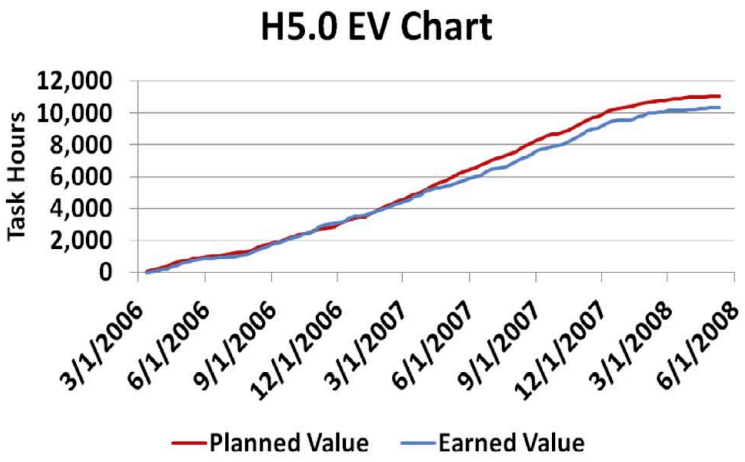


Figure 10: H5.0 Schedule Accuracy

half of the project is due to the delay of several tasks that were not related directly to the product development. These efforts include non-product documentation, post-mortem data analysis, and other non-block related tasks. The team now had an established reliable estimating methodology and it was beginning to show.

Under the H6.0 development effort, the team continued to refine its estimating process. The planned versus actual earned value chart is shown in Figure 11. For 31 months, the software team was able to accrue earned value very consistently with the expected planned value. Although it had taken several blocks, this is the type of planning and execution that the team had hoped for and had finally achieved.

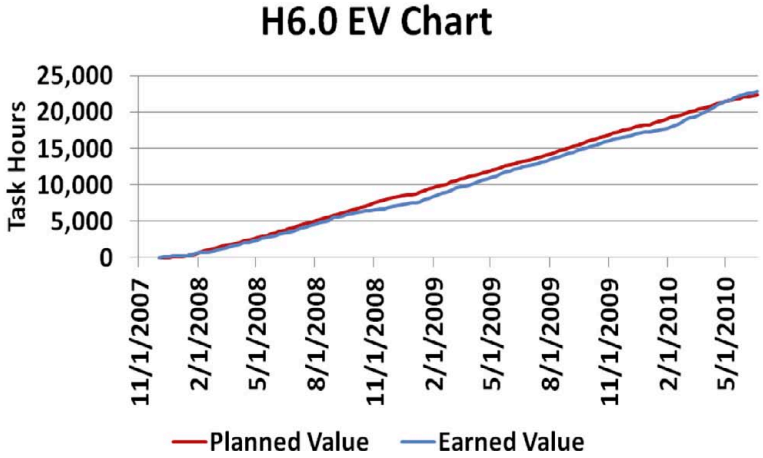


Figure 11: H6.0 Schedule Accuracy

Summary

The team's approach in estimating has enabled it to produce a realistic plan that the team, its customers, and its management are able to effectively use. Even though the team is now able to accurately produce a plan from established estimations, it continues to look for ways to improve its estimating ability because, in the end, it all begins with quality estimates.

Disclaimer:

®CMM is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
SMPSP and SMTSP are service marks of Carnegie Mellon University.

REFERENCES

1. Rickets, Chris A, "A TSP Software Maintenance Life Cycle", CrossTalk, March, 2005.
2. Koch, Alan S, "TSP Can Be the Building blocks for CMMI", CrossTalk, March, 2005.
3. Hodgins, Brad, Rickets, Chris, Lindeman, Robert "How TSP Implementation Has Evolved at AV-8B." TSP Symposium September 2007.
4. Grojean, Carol A., "Microsoft's IT Organization Uses PSP/TSP to Achieve Engineering Excellence", CrossTalk, March, 2005.
5. McConnell, Steve, "CODE Complete", Microsoft Press, 2004.
6. Jones, Capers, "Software Assessments, Benchmarks, and Best Practices", Addison-Wesley Professional, April 2000.

NOTES

1. At the AV-8B Joint System Support Activity, the systems engineering team is responsible for system and software requirements.
2. Process Dashboard is a software planning and tracking tool by Tuma Solutions, LLC.

ABOUT THE AUTHORS

Robert Sinclair is a senior engineer at the Naval Air Warfare Center in China Lake, Ca. He has more than 20 years of experience in developing software in various capacities for the Air Force and the Navy. Mr. Sinclair is currently the supervisor of a group that is developing embedded software. He holds a bachelor's degree in Electrical Engineering from Iowa State University and a Master's in System Engineering from Virginia Tech.

Robert Sinclair
NAWCWD
414600D MS 2016
507 E. Corsair Street
China Lake, CA 93555-6110
760-939-6989

Brad Hodgins is an interim TSP Mentor Coach, SEI-Authorized TSP Coach, SEI-Certified PSP/TSP Instructor, and SEI-Certified Software Developer for the Process Resource Team of the Engineering Division of the Naval Air Systems Command (NAVAIR) located at China Lake, California. Hodgins has been with NAVAIR for 27 years. He has over 20 years experience as a software engineer developing simulation and avionics software. He has been applying PSP/TSP for over eleven years and has coached over 50 TSP/TPI launches. Hodgins earned a BS in Computer Science from California State University, Chico.

Brad Hodgins
NAWCWD
414600 MS 6308
9100 N. Knox Road, Bldg. 01494
China Lake, CA 93555-6110
Phone: 760-939-0666

Chris Rickets is a senior engineer at the Naval Air Warfare Center, China Lake, Ca. He has more than 26 years of experience in developing embedded software for the Navy. Mr. Rickets has been involved in Harrier block upgrades H2.0 – H6.0 and is currently the Mission Systems Computer Technical Lead for the H6.1 block effort and the supervisor for the software developing team. Mr. Rickets holds both a BS and MS in Computer Science from California State University, Chico.

Chris Rickets
NAWCWD
414300D MS 2016
507 E. Corsair St.
China Lake, CA 93555-6100
Phone: 760-939-5838

Challenges in Deploying Static Analysis Tools

Piyush Jain, Infosys Technologies Ltd

DTV Ramakrishna Rao, Infosys Technologies Ltd

Sathyanand Balan, Infosys Technologies Ltd

Abstract. For higher quality software and competitive products, many projects are feverishly deploying static analysis tools. Unfortunately, it turns out that many of the deployments are failures. Some have discontinued static analysis tools altogether. Some continue to use them, but find that the results are not as effective as they hoped.

There are many challenges facing static analysis tool deployments. Although static analysis tools have some weaknesses, the main challenge stems from people. Whether the tool deployment succeeds or fails depends on the people behind it. What are the challenges facing static analysis tool deployments and how can those challenges be overcome? This paper tries to answer that question based on our own deployment of the tools, consultancies with other organizations, and others' experiences.

Introduction

Various studies (e.g., [1]) suggest that around 40% of all software defects could have been detected using Automated Static Analysis (ASA)[1] tools. ASAs are also supposed to help in reducing field failures and time to market. Accordingly, many defense and non-defense projects are increasingly deploying [2] [3] ASAs such as Polyspace [4] and Prevent [5].

Are the deployments successful? Unfortunately, it turns out that many of the deployments are failures. Some projects discontinued ASA altogether. Some continue to use them, but find that the results are not as effective as they hoped. We consider both situations as failures of ASA deployment. These failures stem from the challenges facing ASA deployments. The first situation is more of a "hard failure" where because of the challenges facing ASA deployments, some were discontinued ASA altogether. In the case of the second situation, the failure is a "soft failure" and they continue to use ASA, but because of unmet challenges, the results from the deployments are not as effective as anticipated.

It is important to study the reasons for these failures and the challenges facing ASA deployments. It will help to learn from the mistakes of others in deploying ASA. Second, many are deploying ASA for competitive advantage. Hence, it is important to avoid failure of the deployment.

ASAs have some weaknesses [6] [7], but the leading cause of the failures is ill adoption of the tool by the people in the project. As we will see, just as people are the cause of failure, they are also the solution to make ASA deployments succeed.

This paper is organized in terms of various stages of introducing ASA into a typical project. In each stage, after pointing out how ASA is integrated with that stage, we discuss the challenges faced and how to overcome them. The first four sections –software defense application (Section 1), motivating the stakeholders (Section 2), training (Section 3), and integration into the process (Section 4)–are a prelude to actual use of ASA. The next two sections describe the actual use of ASA by individual developers (Section 5) and at build-time (Section 6). Section 7 adds a feedback stage to tune ASA. Section 8 concludes.

Section 1: Software Defense Application

The defense industry–as shown by projects such as Software Assurance Metrics and Tool Evaluation–is paying significant attention to static analysis tools [3]. This paper helps DoD decision-makers, project managers, and developers in deploying static analysis tools to meet their quality requirements.

Section 2: Motivating the Stakeholders

It is not sufficient to procure the ASA and hope that it will be used. All the stakeholders–developers, middle management, and higher management–need to be motivated. Wide acceptance of ASA by the stakeholders is a prerequisite to get the best possible benefits [8].

In reality, some projects do a poor job in motivating the stakeholders. Lichter et al. [8] found that their ASA deployment was not as effective as expected because they did not spend enough time motivating all of the people involved with ASA.

It is difficult to introduce ASA top down, i.e., only driven by management decision [8]. To get the stakeholders support, they need to be convinced that ASA is important and they benefit from it. For example, higher management demands return on investment analysis, which is a challenge.

Quality Consciousness

ASA is primarily targeted at improving product quality. ASA deployment suffers if the project is not quality conscious.

Do you have quality goals? Do you reward developers for meeting deadlines at the expense of quality? Do developers compete on the basis of fewest defects? Do teams compete with each other for the fewest defects?

We all know the famous saying that goes, "What gets measured gets done." If the project is lacking in quality consciousness, the first step would be to institute some metrics aimed at quality. Here are some metrics worth considering:

- Time to reach system test phase
- Number of defects discovered in the system test phase and their distribution across teams
- Time to reach delivery of system to customers
- Number of defects discovered in the field and their distribution across teams

Once the quality metrics are instituted, the next step would be to suggest ASA as a mechanism to improve the team's performance on these metrics.

Demonstrating the Benefits of ASA

Not everyone in the team will be convinced that ASA will lead to improvement of the team's performance on its quality goals [8]. Demonstrating the benefits of ASA using a case study would convince most.

The case study should show that ASA will be effective at detecting bugs typically noticed in system test and field. It must also show that ASA can detect those bugs with less cost and time compared to the current approaches used in the project. An example of such a case study conducted in industry is reported by Baca et al [9]. We recommend that case study as a model worth emulating for others.

Need for a Champion

Many ASA deployments wither away over time because of a lack of developer and management support. There is a need for a champion who takes the responsibility to include ASA in the development process, and who makes sure that it is sustained. For example, while deploying an ASA [10], Microsoft put this strategy in practice and benefited from it.

Section 3: Training

From their experience in deploying ASA, Lichter et al. [8] conclude, "You need sufficient theoretical and technical know-how to apply ASA systematically." Interactions with ASA require expertise and defect consciousness [11] on the part of developers. Expertise is especially required in (1) configuring ASA, (2) triaging, (3) extending ASA, (4) underlying technology of ASA, like data flow analysis and control flow analysis, (5) how to write code so that it is easily analyzable by ASA, and (6) the internal algorithms used by ASA. Developers need to be trained to gain the expertise.

Although some projects do provide training on the use of ASA, it is found to be incomplete. Some managers are not even aware of the extensive training that is required. The challenge here is that often managers find it difficult to arrange for training in these areas either because of a lack of trainers or because of a lack of information. The training challenges may be met through a Center of Excellence (CoE).

The organization should establish a CoE focusing on ASAs. The objective is to have a single point offering the knowledge that is required to use ASA. For example, the software dependability design group at Nortel [12] works with development teams to train and to include ASA in their development process. A CoE will also be useful for evangelizing ASA.

Section 4: Integration Into the Process

As the old proverb goes, "Failing to plan is planning to fail." Many projects simply drop the ASA into the project and hope that it will show its benefits. The single most likely reason why many ASA deployments fail is that the ASA is not properly integrated into the development process [9].

Introducing a new breed of tool into the development process is easier said than done.

"To be successful, the new tool must fit smoothly into the existing process–it has to make a difference but not cause such a disruption that it is perceived as a source of busy work rather than the solution to a thorny set of problems" [13].

We have observed that management tends to underestimate the effort and cost (direct and indirect) of integrating ASA into the development process–when the reality dawns, ASA suffers or gets neglected.

Approaches to Integration

ASA is a form of Quality Assurance (QA). Studies (e.g., [14] [15]) reveal that ASA complements and does not replace existing QA activities like reviews and testing. Integrating ASA into a development process and combining all the QA techniques to get the best of them in the least time possible remains a challenge. Software development projects have tried two approaches and their variations [16]:

1. Running ASA by a dedicated team: Although tried by many projects, a major challenge with this approach is scalability (as, for example, found at Google [16]).
2. Running ASA by individual developers.

Section 5: Running ASA by Individual Developers

In this approach, developers apply the ASA as part of their regular work on a feature or a defect. ASA will be more effective when applied this way. However, it majorly affects many steps of the development process.

After finishing implementation, the developer runs the ASA, weeds out false positives, fixes the real defects, unit tests the changes, submits the changes for peer review, and checks-in the changes. This section considers how ASA is integrated with these steps of the development process, and what challenges await the project management.

5.1 Estimations

To apply ASA, significant time and effort is required by developers. Do your schedules take that into account? Many do not. It is a difficult problem to update estimation models to take ASA into account. Projects should collect metrics to evolve estimation models (see Section 7.1).

If sufficient time is not set aside for ASA application[2], ASA deployment suffers because it conflicts with the deadlines imposed on developers for their work.

5.2 Triaging and Fixing

After ASA runs, it produces a set of defect reports. The developer has to go through each report to separate true defects from false positives–a process called triaging. Once false positives are weeded out, the developer needs to fix the true defects.

Projects vary in how developers triage and fix. In some projects, developers need not fix immediately, but can open a new defect report containing the issues reported by ASA for future triaging or fixing [17]. Often this is done because estimates do not set aside time for ASA or because of too many false positives. We believe this is a wrong strategy and sets a seed for failure of ASA.

On the other hand, some projects mandate that developers must handle all the ASA-reported issues before check-in. If such a mandate is given–especially without setting aside time–it leads to a different problem. Developers may label real issues as false positives or opt for quick fixes [18] just to keep ASA quiet.

For example, developers are known to simply add a NULL check if the ASA reports a NULL dereference issue, although that is not always the appropriate fix.

5.3 Peer Reviews

After unit testing, developers submit the code for peer review. Not all projects have peer review in their development process. It plays an important role in ASA adoption.

For ASA, peer review serves two purposes: (1) Deal with the challenge of making sure that developers applied and handled ASA reports correctly; (2) Detect those defects missed by ASA.

The review package from developer to reviewers should include among others: How was ASA configured? What were the results? How were the ASA defect reports handled? From this package, the reviewers should verify that false positives are indeed false positives, identified defects are fixed properly, and whether ASA should be altered to find more defects, etc.

Based on that review, the reviewers need to focus on detecting those defects missed by ASA. To do that, they need to have a good idea of the strengths and limitations of ASA–which is often where they are lacking.

5.4 Human Factors

As Gerald Weinberg says, "No matter how it looks at first, it is always a people problem."

For most projects, project sociology is more important than technology [19]. This applies to ASA use also. ASA is considered a pure technology, where as it has numerous sociological angles. Managers tend to focus on technology and not on the sociology side [20] with the resultant failure of ASA deployments.

In this section, we consider the role developers and reviewers play in the success or failure of ASA and the challenges they present to management.

5.4.1 Changing Habits is Hard

The main barrier to adoption of ASA lies in the ability of developers to impose the discipline required to make ASA a routine part of their work [17].

Introducing ASA is a change for most developers. Changing habits is hard [13]. Some regard ASA as a nuisance.

Some ASAs are initially difficult to use, but over time developers may start appreciating them. This has been confirmed experimentally [4]: The experimenters created three versions of code with different errors. For model checking, false positives decreased across all three versions because of developers creating better abstractions as the experiment proceeded.

In practice, many developers give up before they come to a stage where they appreciate ASA. Process methodology and project management need to ensure that developers persist with ASA. Management should convince developers that ASA ameliorates the frustrating debugging associated with field failures.

5.4.2 Lack of Motivation

Some projects deploy ASAs ad hoc–they depend upon developer's motivation in using ASA [9]. Some deploy ASA in a planned fashion, but they still leave it to the developer to decide what warnings are important and what are not, etc. Again, a lot of responsibility is with the individual developer.

When so much responsibility is with a developer, whether ASA works or not depends on the developer's motivation. A developer's motivation depends on many factors. Do you feel bad about defects in your code? Developers differ. Where developers feel proud of their work, the possibility that they successfully deploy ASA is high, and vice versa. To ensure that ASA is used by developers, development process should include a rule like no code can be checked in until ASA results pass a set criterion. The criterion could be no defects, no severe defects, or fewer than five minor defects, etc.

Team organization and project policies also have a role in the developer's motivation of using ASA effectively. A specific situation worth considering is the separation sometimes seen between two groups of development: feature development and sustenance. The development group introduces new features and sustenance group fixes defects. For effective ASA, feature developers need to spend time using ASA. But often the policies and goals of development group are such that they are indirectly discouraged to use ASA. Some organizations [21] have the policy where the developer of a feature is responsible for fixing the defects in the feature. If a feature developer is responsible for fixing the defects, then the developer will have to spend time to debug and resolve the testing-reported defects. It will hinder the developer in moving to work on new features. Since ASA-reported defects can be fixed sooner than testing-reported defects [22], developers would prefer to use ASA to shorten their debugging cycles for resolving testing-reported defects.

5.4.3 Reducing Discipline Because of ASA

Because ASA is there to detect defects, will it lead to sloppy coding by developers and less effective review by reviewers? All too often a pre-tested module does not get inspected properly, "Well, that [module] works OK [23]. Why waste time inspecting it?" The situation is analogous with ASA. There is anecdotal evidence that ASA presence leads to some loss of discipline in developers and reviewers (based on our interaction with some project managers). Private self-assessment (see Section 7.2) would help in mitigating this challenge.

Section 6: Build-time Running of the Tool

To complement developers running the tool, periodical (e.g., weekly) ASA is run on the entire codebase. Not all projects have this important step of build-time running of the tool. In this step, the ASA is configured to do deeper analysis compared to developers running the tool. A central team triages the defect reports and opens defects in defect tracker for later fixing by developers.

The results of build-time running ASA are also useful for generating reports and metrics. Although ASAs provide various types of reports, a major challenge is providing reports management can understand and relate to. The reports should show trends of number of defects and defect types across builds. Wherever possible, the reports should correlate ASA-reported defects to defects that managers and users are aware of. For example, maybe an ASA-reported defect could explain a field failure of the system. Such reports enhance the support of both management

and developers to ASA adoption and helps sustain it.

The major challenge in this stage is not fixing the ASA-reported defects. As mentioned earlier (Section 5.2), other stages in the development process also contribute to this challenge.

Not Fixing the ASA-reported Defects

You will get the real benefit from ASA only if ASA-reported defects (after pruning false positives) are fixed [24]. In many projects, the emphasis on fixing ASA-reported defects is minimal; hence they tend not to get fixed [25]. It negates the whole purpose of deploying ASA. We observed that some of the unfixed ASA-reported defects actually turned up in field.

Here are some reasons that contribute to non-fixing of ASA-reported defects [7]:

Delayed detection of ASA defects: The later an ASA defect is detected, the lesser the chance that it will be fixed. That is why ASA defects detected at build-time or by a dedicated team have a lesser chance of getting fixed compared to defects detected by an individual developer.

Allowing ASA defects to accumulate: In some projects, developers do not have to triage or fix the ASA-reported defects prior to check in (see Section 5.2). This leads to accumulation of ASA defects and reduces the probability of fixing them.

Severity and priority of ASA defects not clear: Testing reported defects normally shows that something is broken, and from that it can be ascertained how urgent the fix is (priority) and the consequences of not fixing it (severity). ASA defects normally do not show that.

This challenge can be dealt with in the following ways. First, developers should be encouraged and facilitated to fix the ASA-reported defects before check in rather than deferring them. Second, management should strive for nightly ASA builds rather than weekly builds, although it might mean infrastructure needs to be heavily upgraded for a faster ASA run. Management and developers should determine the right frequency. Third, when the ASA finds a defect in the nightly build, it should be integrated with source code management system to find which recent check in is responsible for this issue. Then it can immediately inform the associated developer for faster and easier fixing.

Section 7: Retrospectives

Many–but not all–projects have retrospectives, where they look back on the project. Retrospectives are important for effective ASA. They help in tuning of ASA and the process.

7.1 Tuning ASA and the Process

Based on the retrospective, ASA should be tuned for detecting new types of defects and for higher performance. The development process needs to be tuned. For example, estimation models need to be tuned to set aside time to use ASA by developers in various activities like triaging and fixing. To do the tuning, the development process should collect related metrics.

7.2 Appraisals

Appraisal of developers is common in software organizations. ASA can provide additional means to appraise. If not used properly, it can backfire. The balancing act is a challenge.

ASA provides many metrics and reports (see Section 6). Many of them relate to defects. The temptation to use them for measuring developer's programming capabilities is high [8]. That is not a good idea [19]. It can easily upset the whole program of deploying ASA and will be counterproductive [8]. Developers either find ways to bypass ASA or apply a coding style that only leads to acceptable results of ASA but not result in overall product quality [8].

Defect counts and their categorization are especially useful for developers. While they should not be used to measure developers' capabilities, it is important to deploy such metrics for private self-assessment [19] and self-learning only. Some projects do not provide such metrics for private use. Instead, they tend to deploy metrics for management use only.

Section 8: Conclusion

ASA is an important tool in the quest towards higher quality. But its deployment is not easy. The challenges include:

- Motivation (e.g., not motivating the stakeholders on why the tool is important and how they are going to benefit from it);
- Training (e.g., inadequate training on the technology that underlies the tool and not raising the defect consciousness of developers);
- Development process (e.g., difficulties in integrating the tool into development process);
- Developers (e.g., developers resenting the additional over head of the tool);
- Project management (e.g., difficulties in scheduling to fix the tool reported defects);
- Performance appraisals (e.g., team dissatisfaction because of appraising engineers using metrics generated by the tool).

Forewarned is forearmed. By being aware of the challenges one may face in deploying ASA beforehand, one can be prepared to deal with them.

Acknowledgments:

We thank Ilan Kumaran, Thomas George, Sujay Gupta, Ravi Kumar B, and Srikanth S for sharing their experiences of ASA deployments and their comments on earlier versions of this paper. The paper particularly benefited from the insightful and detailed comments from the CrossTalk Editorial Board.

Disclaimer:

The statements and opinions expressed in the paper are authors' own and not their employer's.

REFERENCES

1. L Hatton. Software failures-follies and fallacies. IEE Review, 43(2):49–52, 1997.
2. A. German. Software static code analysis lessons learned. CrossTalk: The Journal of Defense Software Engineering, November 2003.
3. Yannick Moy. Static Analysis Is Not Just for Finding Bugs. CrossTalk: The Journal of Defense Software Engineering, September 2010.
4. G. Brat et al. Experimental evaluation of verification and validation tools on Martian Rover software. Formal Methods in System Design, 25(2):167–198, 2004.
5. A. Bessey et al. A few billion lines of code later: using static analysis to find bugs in the real world. Comm. of the ACM, 53(2):66–75, 2010.
6. Paul Black. Static Analyzers in Software Engineering. CrossTalk: The Journal of Defense Software Engineering, March 2009.
7. D T V Ramakrishna Rao, Piyush Jain, and Sathyanand Balan. Why Static Analysis Tool Deploy ments Fail? (And How...). In Embedded Systems Conference (ESC), Bangalore, July 2010.
8. H. Lichter and G. Riedinger. Improving software quality by static program analysis. Software Process: Improvement and Practice, 3(4):235– 241, 1998.
9. Dejan Baca. Automated static code analysis – A tool for early vulnerability detection. PhD thesis, Bleking Institute of Technology, Sweden, 2009.
10. T. Ball et al. SLAM and Static Driver Verifier: Technology transfer of formal methods inside Microsoft. LNCS, 2999:1–20, 2004.
11. D T V Ramakrishna Rao. Defect Detection By Developers. CrossTalk: The Journal of Defense Software Engineering, pages 8–11, March 2009.
12. J. Zheng et al. On the value of static analysis for fault detection in software. IEEE Transactions on Software Engineering, 32(4):240–253, 2006.
13. P. Chandra et al. Putting the tools to work: How to succeed with source code analysis. IEEE security & privacy, 4(3):80–83, 2006.
14. S. Wagner et al. Comparing Bug Finding Tools with Reviews and Tests. In Proc. 17th Int'l Conf. on Testing of Communicating Systems (TestCom05), pages 40–55. Springer, 2005.
15. J. Nazario. Source code scanners for better code. Linux Journal, January 2002.
16. N. Ayewah et al. Using static analysis to find bugs. IEEE software, 25(5):22–29, 2008.
17. Melissa Webster. The Software Quality Imperative. Technical report, IDC, 2005.
18. Russel King. "Do not misuse Coverity please".
19. Tom Demarco. Why does software cost so much? Dorset House Publishing, 1995.
20. T. DeMarco and T. Lister. Peopleware: productive projects and teams. Dorset House Publishing Company, Incorporated, 2nd edition, 1999.
21. S. Maguire. Debugging the development process. Microsoft Press, 1994.
22. Brian Chess and Gary McGraw. Static analysis for security. IEEE Security and Privacy, 2(6):76–79, 2004.
23. J.G. Ganssle. The art of designing embedded systems. Newnes, 2000.
24. Matthew Hayward. The Truth Behind Static Analysis Pitfalls. EE Times, <http://www.eetimes.com/design/embedded/4008210/TheTruth-Behind-Static-Analysis-Pitfalls> January 2009.
25. William Pugh. Making Static Analysis Part of Your Build Process. Presentation to the Silicon Valley Java Users Group, 2009.

NOTES

1. In this paper, we use the terms "The Tool" and "ASA" interchangeably.
2. Even if developers take additional time because of ASA, overall product release time still tends to improve because of reduction in testing and debugging cycles.

ABOUT THE AUTHORS

Piyush Jain, PMP, is a Delivery Manager in Product Engineering Unit of Infosys Technologies Ltd, Bangalore, India. He has more than 17 years of experience in software development. In his role of delivery manager he is responsible for new business development and end-to-end delivery management of projects for global customers. He has provided consulting services to leading networking firms on how to improve engineering efficiencies and effectiveness of tools deployment in product development and testing. Prior to taking up management role, he has had extensive experience in embedded development on networking products with primary focus on system engineering and L3 and L4 protocol development. He is PMP certified and has published papers in PMI India conference and other conferences/forums. He holds a BE in Computer Engineering from National Institute of Technology (NIT), Surat.

Infosys Technologies Ltd.
Electronics City,
Hosur Road,
Bangalore – 560100
Phone: +918041166289
Fax: +918028521695
Mobile: +919880596947
E-mail: Piyushj@infosys.com

D.T.V. Ramakrishna Rao is a Senior Technology Architect in the Product Engineering Division of Infosys Technologies Ltd, Bangalore, India. He has 14 years of experience in software development with primary focus on building high-end networking systems. He has presented and published 15 papers on project management, static analysis, networking, and defect management in international journals and conferences including CrossTalk, IEEE, IETF, and PMI. He created two new static analysis tools to detect interfacing bugs and endian bugs. He has evaluated, deployed, and worked with various static analysis tools including Coverity's Prevent, cppcheck, and Sparse. He recently published the paper "Defect Detection by Developers" in Mar/Apr 2009 edition of CrossTalk. He holds a B.Tech in Computer Science from National Institute of Technology (NIT), Warangal, and an M.Tech in Computer Science from Indian Institute of Technology, Kanpur.

Mobile: +919845554451
E-mail: ramakrishnadtv@infosys.com

Sathyanand Balan is a Senior Technology Architect in the Product Engineering Division of Infosys Technologies Ltd, Bangalore, India. He has 11 years of experience in software design and embedded software development for networking systems. He is a lead architect in a large engineering program for a global customer. He was part of the tools group that rolled out a static analysis tool (Coverity Prevent) for a leading networking OEM in their development process. He has conducted various training programs on board bring up, Network processors (Wintegra), embedded system design and development. He holds a B.Tech in Electronics and Communication from NIT, Calicut.

Mobile: +919845392221
E-mail: sathyanandb@infosys.com

Tailoring's Last Name Is Not "Out"

David P. Quinn, MOSAIC Technologies Group, Inc.

Abstract. Frequently, organizations think tailoring is about skipping steps in the organization's standard process. This misinterpretation causes many organizations to backslide in their process maturity and prevents them from gaining great insight into their process potential. Once an organization learns that tailoring's last name is not "out," they mature their process implementation more rapidly.

When first working with an organization on tailoring processes, they always seem anxious to find out what part or parts of their processes they can skip. After all, isn't that what tailoring is about–taking out the unnecessary parts of a process for a particular project? After shaking my head once again, I let them know that tailoring is not about skipping parts of processes but scaling them to a specific need. This consistent misinterpretation of tailoring out parts of processes has several causes that need to be addressed.

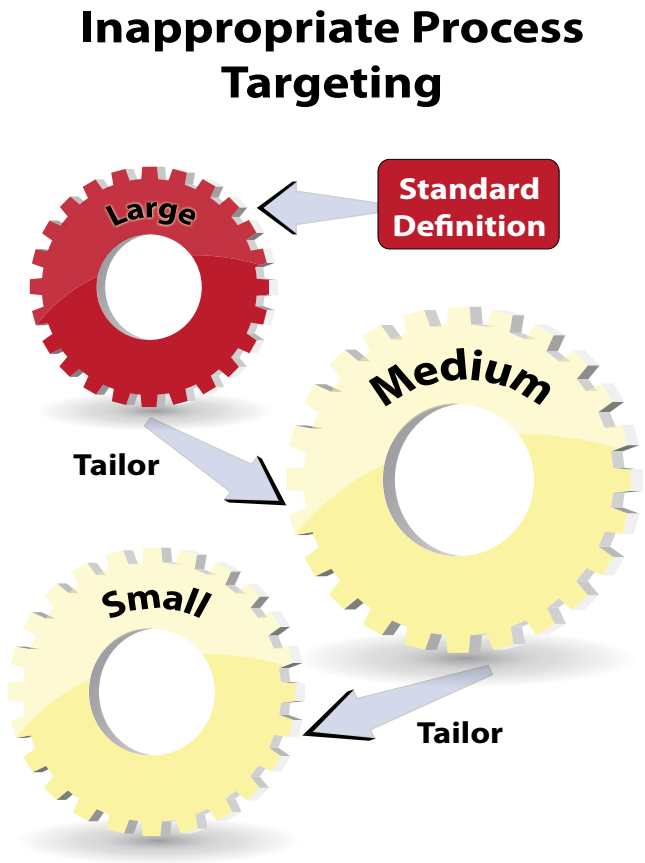
For CMMI®, tailoring processes is a major component/aspect of institutionalizing a "defined process." It is the primary element of Generic Practice 3.1 and what differentiates Integrated Project Management from Project Planning and Project Monitoring and Control. When done correctly, tailoring takes a generic process and makes it meaningful to its users. In other words, it transforms the Organization's Set of Standard Processes into a Project's Defined Process.

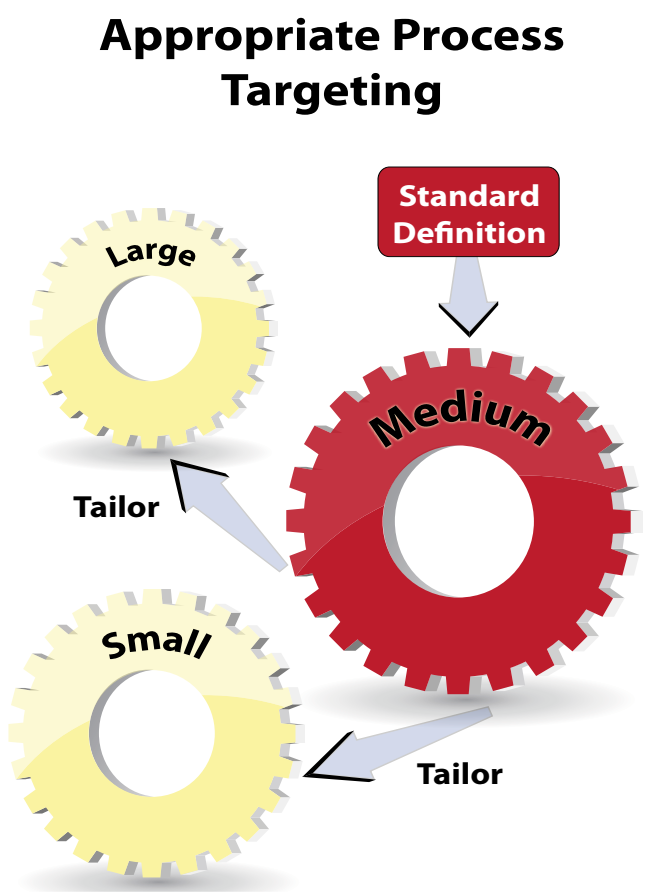
Unfortunately, many people that are new to process tailoring assume that tailoring involves eliminating steps or skipping entire processes. This misinterpretation of tailoring tends to inhibit the potential of the organization and misses the intent of tailoring. Besides making a process meaningful, tailoring should be the source for organizational learning.

I generally find that the usual root cause of the misinterpretation of tailoring is an organization's assumption that they should have a single process. Let's face it, a "one size fits all" solution usually fits no one. That is why people are anxious to skip parts or all of a process when addressing tailoring.

In order to position itself for tailoring, the organization needs to examine its approach to process definition. It always fascinates me when organizations realize they have different types of projects but then target the wrong project type to document their processes. For instance, most organizations will divide their projects into large, medium, and small based on staff size or number of estimated hours. They will document their processes for the large projects and assume they can tailor the processes down for the other project types.

The problem with this approach is that these large projects usually are the exception and not the norm for the organization. It does not make sense to define processes that represent the least common work for the organization. Organizations need to





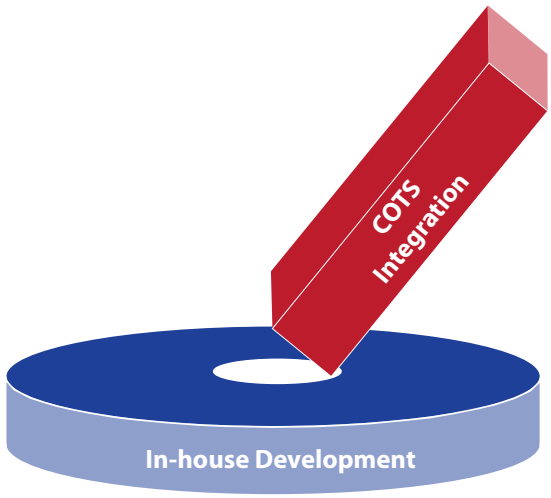
document processes for the "sweet spot," where most of the work is performed. If a medium-sized project represents most of the organization's work, the process should be defined for that project type. The process then can be tailored up or down for the other project types.

This brings us to another key aspect of tailoring. Tailoring is really about scaling and scoping, not skipping. Organizations can tailor up or down, not out. When a project needs more process detail, more detail is added. When less detail is needed, the process can be summarized, not skipped. For instance, projects should always have physical configuration audits. The standard process may be designed for small projects (one or two person projects lasting a month or two) since that is the major project type for the organization. The standard process may state that a physical configuration audit is conducted right before delivery. Tailoring for large and medium sized projects may specify that physical configuration audits are added at the end of unit testing. Tailoring for large projects may require that physical configuration audits are conducted at the end of each major milestone. This is tailoring up for projects, not just tailoring down (and definitely not out).

Appropriate tailoring needs to happen during process definition. The organization should identify tailoring factors such as project size, product life cycle (e.g., new development, maintenance, acquisition), customer type (e.g., federal government, local government, commercial), life cycle model (e.g., waterfall, spiral, Scrum), etc. These tailoring factors and their instructions are what create tailoring guidelines as specified in Organizational Process Definition Specific Practice 1.3.

Other input for tailoring should be the waiver process. When a project feels the need to do something different from the organizational standard set of processes, the organization should monitor this change in order to learn from what is done. The waiver should specify not only what they will not be doing from the standard organization process but what they will do differently so the organization has a potential alternative practice. What a project that has a waived process does may identify additional tailoring factors or additional process options.

Finally, the organization needs to recognize when it is beyond tailoring and into creating another process for the organization to add to its set of organizational standard processes. I worked with a group one time that had three organizational standard processes to choose from: development, maintenance, and databases. The group I worked with did COTS integration projects. They would brute force their projects into the development process by significantly tailoring the process. Essentially, they were trying to force the proverbial square peg into a round hole. When I pointed out that they had enough experience at tailoring the process that they should define a fourth standard process type, they responded with, "We can do that?" Consistent tailoring should either result in a process change or a new standard process.



When preparing to tackle process tailoring, an organization should keep several thoughts in mind. First, the organization needs to shoot for the "sweet spot" when defining processes. The organization needs to identify tailoring factors when defining the process. The factors could come from the waiver process and eventually lead to a new standard process for the organization. Based on the process definition and tailoring factors, projects will be scaling and scoping the process in order to tailor it to their specific needs. All this leads back to the primary thought; tailoring's last name is not "out."

Disclaimer

®CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

David P. Quinn is the Director for Process Services at MOSAIC Technologies Group, Inc. He has over 25 years software and systems development, maintenance, and management experience. He has over 15 years experience as a process improvement consultant, helping large financial firms, defense contractors, telecommunications companies, aerospace contractors, and healthcare providers. He is certified by the Software Engineering Institute as a SCAMPISM Lead Appraiser and Instructor for CMMI for Development and CMMI for Services.

MOSAIC Technologies Group, Inc.
8161 Maple Lawn Blvd, Suite 430
Fulton, MD 20759
Phone: 301-725-0925 x724
Fax: 301-725-0895
E-mail: dquinn@mosaicsgroup.com

On Becoming a Software Engineer

Once there existed a newly graduated engineer named Fred. Above all else, Fred had come to appreciate the beauty in writing elegant and efficient code. Knowing that he lacked wisdom, he approached a wise master for advice. "Oh Wise Master," Fred asked, "How may I become a software engineer?"The Wise Master looked deep into the heart of Fred, and saw his sincerity. "There are three tasks you must complete. If you are willing to undertake these tasks, I can teach you. Are you willing?" "Yes, Oh Wise Master. I yearn to gain wisdom." "Very well," the Wise Master said. "You must first master the Morass of Meetings. This is your first task."

Fred first attempted to slay the Morass. He tried to shorten and even kill them. However, all was for naught. As he tried to shorten meetings, they inexplicably became longer. When he tried to kill the meetings, much like a Hydra, two sprang up from each one that was killed. Fred began to realize that each meeting had a purpose, and until the purpose was met, the meeting could not be slain. In time, Fred learned to embrace meetings, for in meetings, kernels of wisdom could be found. He learned to balance joint application design sessions, preliminary design reviews, critical design reviews, user acceptance meetings, and architectural reviews. He spent much time using e-mail to organize and arrange meetings learning to respond in a timely fashion to all manner of unreasonable requests.

After Fred mastered the Morass, he returned to the Wise Master. "Oh Wise Master," he said, "I have learned to glean understanding from the Morass of Meeting, and discern wisdom from folly. I have found balance in scheduling meetings back-to-back, and I have even learned to fight the dreaded creeping waistline found in each meeting's den of doughnuts. I am ready for the next task." The Wise Master saw that indeed Fred had mastered the Morass, and replied, "Your second task is to tame the Tangle of Incomplete User Requirements."

Fred labored long and hard to simply uncover the Tangle, for many user requirements were frequently obscure and tried to hide from the light of day. Many were not apparent, and required discernment to uncover. Nevertheless, Fred persevered and by virtue of hard work and long hours was able to tame the Tangle of Incomplete and Inconsistent User Requirements. He was proud of his achievement, and returned to the Wise Master to demonstrate his prowess.

The Wise Master was impressed, but cautioned Fred saying, "You have indeed mastered the Morass of Meetings and tamed the Tangle of Incomplete and Inconsistent User Requirements. However, these two tasks were to give you courage so that you would not be faint of heart for the last task: Juggling Conflicting and Changing Requirements and Priorities." Fred trembled at these words, but did not deter from his mission.

Fred found out that every class of user had conflicting priorities. Every user had changing requirements. Fred also discovered, much to his dismay, that often the budget and schedule were also conflicting. Even the very management that first set him forth in his labor often changed their priorities daily. Fred initially set out to make all users happy, but soon realized that making one happy often made all the others unhappy. Each user had a specific priority, and each priority seemed in direct contrast to all others. Fred eventually realized that no one user would ever be totally happy–the best that could be accomplished was to try and not make any one user totally unhappy. This "not unhappy" concept filled him with wonder, as he learned to tradeoff "not unhappiness" in one class of user for more "not unhappiness" in another. His efforts were herculean, but eventually he no longer feared for his life when confronting users. He learned that when requirements and priorities changed, he needed to arrange more meetings and fight the demons of incompleteness and inconsistency. In times of great despair, he found solace in a magic elixir of coffee, antacids, beta blockers, and aspirin.

At long last, Fred felt happy with his progress, and returned to the Wise Master. "Oh Wise Master," he exclaimed, "I have succeeded in the three tasks that you laid out for me. I have learned to master the Morass of Meetings. I fought long and hard, and by dint of effort, managed to tame the Tangle of Incomplete and Inconsistent User Requirements. And–by the sweat of my brow and gnashing of teeth–I have even discovered the path to Juggling Conflicting and Changing Requirements and Priorities. I have demonstrated my worthiness in accomplishing all of these tasks. Now, oh great master, I am ready! At last, impart to me your secrets of becoming a software engineer!"

"Why Fred," the Wise Master exclaimed, "do you not see? You have been a software engineer all along! As you have accomplished each of these three tasks, you have found your true path to becoming a great software engineer!"

"But," Fred interrupted, "I have had no time to write code. When do I write code?"

"Code?" the Wise Master thundered in anger. "Oh, you want to write code? Why didn't you say so? That's totally different. It has nothing to do with the tasks I gave you!"

David A. Cook, Ph.D.
Stephen F. Austin State University
E-mail: cookda@sfasu.edu

Shamelessly adapted from an idea I found at <http://www.jokes4teachers.com/J0145.php> (no author was given).